LoRA on the Go: Instance-level Dynamic LoRA Selection and Merging

Anonymous ACL submission

Abstract

Low-Rank Adaptation (LoRA) has emerged as a parameter-efficient approach for fine-tuning large language models. However, conventional LoRA adapters are typically trained for a single task, limiting their applicability in real-world settings where inputs may span diverse and unpredictable domains. At inference time, existing approaches combine multiple LoRAs for improving performance on diverse tasks, while usually requiring labeled data or additional task-specific training, which is expensive at scale.

In this work, we introduce LoRA on the Go (LoGo), a training-free framework that dynamically selects and merges adapters at the instance level without any additional requirements. LoGo leverages signals extracted from a single forward pass through LoRA adapters, to identify the most relevant adapters and determine their contributions on-the-fly. Across 5 NLP benchmarks, 27 datasets, and 3 model families, LoGo outperforms training-based baselines on some tasks upto a margin of 3.6% while remaining competitive on other tasks and maintaining inference throughput, highlighting its effectiveness and practicality.

1 Introduction

Recent advances in large language models (LLMs) such as Llama (Dubey et al., 2024) and Deepseek (Bi et al., 2024) have led to remarkable progress across diverse natural language processing (NLP) tasks. While these models demonstrate strong generalization capabilities, achieving state-of-the-art results in specialized domains often requires task-specific fine-tuning (Wei et al., 2022). However, the massive scale of modern LLMs makes full fine-tuning computationally prohibitive, motivating research on Parameter-Efficient Fine-Tuning (PEFT) methods that adapt models by updating only a small subset of parameters (Houlsby et al., 2019; Li and Liang, 2021; Liu et al., 2022).

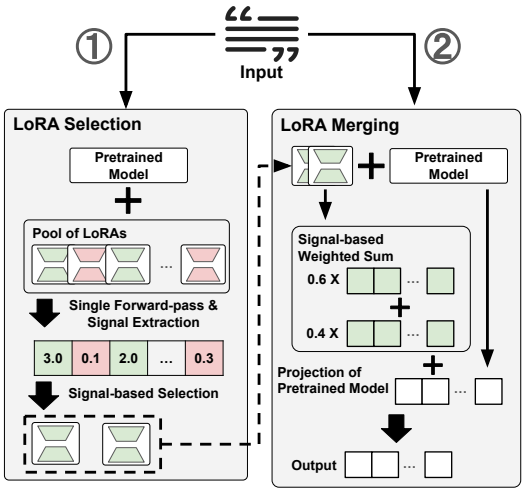


Figure 1: Overall workflow of the proposed LoRA on the Go (LoGo) framework.

Among them, Low-Rank Adaptation (LoRA) (Hu et al., 2022) is particularly effective, introducing trainable low-rank matrices while freezing pre-trained weights, thus reducing trainable parameters without sacrificing performance.

Although LoRA provides an efficient adaptation mechanism, the adaptors are typically optimized for a single domain or task. In contrast, real-world applications increasingly demand generalization to unseen tasks or tasks that require specialization across multiple domains. Recent works (Huang et al., 2024; Zhao et al., 2024) explore the possibility of simultaneously leveraging multiple LoRAs trained on diverse tasks.

Existing multi-LoRA approaches share a key limitation when composing LoRAs together: to first select relevant LoRAs from a pool and then to merge them, they assume well-defined tasks and rely on labeled data. For instance, LoRAHub (Huang et al., 2024) learns fixed composition weights for each new task using a small labeled set, while LoRARetriever (Zhao et al., 2024) trains a retrieval model over mixed-task data but still depends on labeled examples to compute retrieval embeddings. Such dependence on task homogene-

ity and labeled supervision restricts scalability in real-world scenarios, where LoRAs may be continually added or updated and labeled data may be unavailable.

In generic conversational systems such as AI copilots (Microsoft, 2023) or multi-domain assistants (OpenAI, 2023; Gemini, 2023), these assumptions rarely hold. User queries are highly heterogeneous, often privacy-sensitive, and may transition across unrelated tasks (e.g., summarization, reasoning, translation, coding) without explicit task boundaries. Meanwhile, LoRA pools evolve dynamically as new adapters are introduced or deprecated, making task-specific retraining or labeled data collection expensive and impractical.

These challenges motivate our central research question: ***How can we dynamically select suitable LoRAs for each input, given an evolving LoRA pool and heterogeneous tasks, without labeled data or retraining?***

In this work, we introduce **Lo**RA on the **Go** (**LoGo**), a framework that operates without any pre-defined data or retraining assumptions, enabling seamless integration with a dynamic LoRA pool. LoGo adopts an **instance-specific** perspective—selecting and merging LoRAs on the fly for each input. Since selection and merging must occur over many candidates in real time, our method is entirely training-free. The core intuition is that *LoRA activations already encode signals of relevance*: when a LoRA is well-suited to an input, its updates exert stronger influence on model outputs (e.g., inference for WNLI (Levesque et al., 2012) benefits from LoRAs trained on SNLI (Bowman et al., 2015) and MNLI (Williams et al., 2018)).

Building on this, LoGo extracts simple yet informative signals—such as the norm or entropy of LoRA activations—from a single forward pass with all LoRAs attached. These signals are used to identify relevant adapters, which are then merged via a weighted sum of activations, where weights are determined by the extracted signals (see Fig. 1).

We evaluate LoGo on diverse NLP benchmarks including BIG Bench Hard (BBH) (Suzgun et al., 2023), Translation (Bojar et al., 2014, 2016), Struct-to-Text (Gehrmann et al., 2021; Lin et al., 2020; Nan et al., 2021; Novikova et al., 2017; Gardent et al., 2017), Closed-Book QA (Clark et al., 2018; Kwiatkowski et al., 2019; Joshi et al., 2017), and Natural Language Inference (Nie et al., 2020; Wang et al., 2018). We train three model families over 260 FLANv2 tasks (Wei et al., 2022; Chung et al., 2024) and evaluate across five benchmarks spanning 27 datasets.

Results show that LoGo, even without retraining or data assumptions, often surpasses training-based baselines by up to 3.6% on tasks like Struct-to-Text,NLI, while maintaining competitive performance on the rest. LoGo also preserves comparable throughput during selection, merging, and inference. Our analysis confirms that its overhead is amortized in long-output tasks such as summarization or chain-of-thought reasoning, making it highly practical. Across different settings, LoGo shows consistent performance. We will publicly release our code upon acceptance.

Our main contributions are summarized as follows:

- We identify the limitations of existing multiple LoRA-based approaches, which rely on labelled data availability and additional training, making them expensive for real-world deployment.
- We introduce LoRA on the Go (LoGo), a training-free, instance-specific framework that dynamically selects and merges suitable LoRAs for each input using activations extracted in a single forward pass.
- We conduct extensive experiments on 5 standard benchmarks encompassing 27 datasets over 3 model families, showing that LoGo not only outperforms training-based baselines but also has comparable throughput.

2 Related Work

To perform predictions for a new task, a few studies have explored dynamic combination of multiple existing LoRA adapters, each trained on different tasks, based on a given input. For instance, Mixture of LoRAs (MoA) trains a router to select a single LoRA from a pool of candidates (Feng et al., 2024). LoRAHub instead learns task-specific weights to merge the parameters of multiple LoRAs via weighted summation (Huang et al., 2024), while Mixture of LoRA Experts (MoLE) also learns weights but applies them to the outputs of LoRA adapters rather than to their parameters (Wu et al., 2024). All of these methods assume access to labeled samples from the target input distribution and rely on such data to train either the router or the merging weights in a task-specific manner. However, this assumption rarely holds in practice: inputs usually arrive from diverse and unpredictable domains.

More recently, LoRARetriever (Zhao et al., 2024) highlighted the need to address instance-specific scenarios, where inputs are drawn from heterogeneous domains. To this end, they train an auxiliary language model to retrieve the most relevant LoRAs for a given input, using samples from a mixed distribution of datasets. While this approach moves toward an instance-level perspective, it still requires training a large embedding model to compute reliable similarity scores. Inclusion of LoRAs in this framework would require (a) samples from the corresponding datasets, and (b) re-computing an embedding point for the new LoRAs in the existing embedding space. This embedding might not be appropriate for out-of-domain (OOD) scenarios, for e.g. non-English tasks. Alongside, it may also tamper with the in-domain performance due to entangled embedding points with the OOD LoRAs. Hence, the performance of the resulting model may decline when the inputs deviate significantly from the training distribution of the embedding model.

In contrast, LoGo seamlessly integrates incoming LoRAs and adopts the instance-specific perspective that requires no training or additional data samples. By leveraging activations extracted from adapters themselves, LoGo identifies sample-relevant LoRAs on the fly, avoiding reliance on auxiliary models or predefined training samples.

3 The Proposed LoGo Methodology

The goal of LoGo is to dynamically select and merge the most relevant LoRA adapters for each input, without relying on task-specific training. We begin by formalizing the problem setting, where a pretrained backbone LLM is equipped with a pool of LoRA adapters, each providing low-rank updates to projection matrices (Section 3.1). Given a new input, LoGo performs a single forward pass with all adapters attached, extracts their projection outputs from a designated block, and computes signal scores (e.g., norms or inverse entropy) from these projection outputs to measure adapter relevance. The top-scoring adapters are then selected as candidates (Section 3.2). Next, the selected adapters are merged efficiently through a weighted sum of their outputs, where the weights are directly determined by the extracted signals (Section 3.3). This design allows LoGo to adaptively combine multiple LoRAs on the fly, while maintaining real-time efficiency and avoiding any additional training overhead. We provide an algorithm that summarizes overall procedure of LoGo in Appendix B.

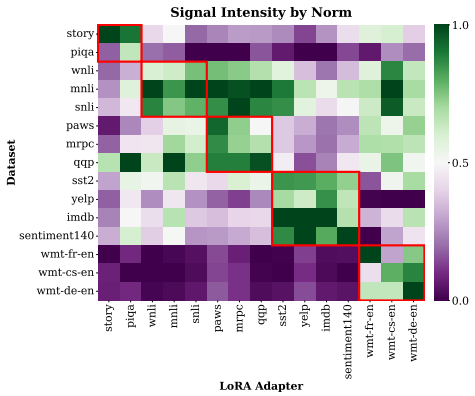


Figure 2: Heatmap illustrating signal patterns across LoRA adapters trained on top of the Qwen-2.5-7B backbone. The x-axis represents LoRAs trained on different tasks, while the y-axis corresponds to datasets from those tasks. Each cell shows the ℓ_2 norm of the projection outputs. The norm values are min-max normalized to [0,1] across datasets for each LoRA. Related task clusters are highlighted in red boxes. More results on signal intensity are in Appendix A.

3.1 Problem Formulation

We consider a setting where a pretrained model f_θ is paired with a set of N LoRA adapters $\mathcal{L} = \{L_i\}_{i=1}^N$, each of which is fine-tuned on a distinct task T_i. Given an input sequence $\mathbf{x} = (x_1, ..., x_P)$ of length P, the model generates an output sequence $\mathbf{y} = (y_{P+1}, ..., y_{P+t})$ of length t.

The pretrained model f_θ consists of M Transformer blocks $\mathcal{B} = \{B_j\}_{j=1}^M$, where each block B_j contains a self-attention mechanism with head-specific query, key, and value projections, and a feed-forward network. We denote the query and value projection matrices of block B_j (we omit head subscripts for sake of clarity) as $\mathbf{W}_j^{(Q)}$ and $\mathbf{W}_j^{(V)}$, respectively. A LoRA adapter $L_i \in \mathcal{L}$ attaches to the projection matrices $\{(\mathbf{W}_j^{(Q)}, \mathbf{W}_j^{(V)})\}_{j=1}^M$ and introduces low-rank updates. For example, for the query projection in block B_j, let $\mathbf{h}_j$ denote the latent input. Then, the adapter L_i produces an update via a low-rank projection $\Delta\mathbf{W}_{i,j}^{(Q)}\mathbf{h}_j$, where $\Delta\mathbf{W}_{i,j}^{(Q)} = \alpha_{i,j}\mathbf{A}_{i,j}\mathbf{B}_{i,j}$. Here, $\mathbf{A}_{i,j}$ and $\mathbf{B}_{i,j}$ are the low-rank matrices of LoRA, and $\alpha_{i,j}$ is a scaling factor.

3.2 Selection of Instance-specific LoRAs

To identify the most relevant LoRAs for a given input, LoGo relies on signals extracted during a single forward pass through the base model f_θ equipped with all available adapters in $\mathcal{L}$. Unlike prior approaches that require additional training,

our method is training-free. The procedure incurs only the cost of one probe computation, making it practical even under real-time constraints.

Formally, we define the *adapter-augmented model* $f_{\theta,\mathcal{L}}$ as the base model f_θ where every adapter $L_i \in \mathcal{L}$ is attached to its designated projection matrices. Let $B_T \in \mathcal{B}$ denote the target block from which we extract signals, and let $\mathbf{h}_T$ be the hidden representation entering block B_T. For each adapter $L_i \in \mathcal{L}$ attached to query projection matrix $\mathbf{W}_T^{(Q)}$ of B_T, we define projection output as

$$\mathbf{o}_{i,T} = \Delta\mathbf{W}_{i,T}^{(Q)}\mathbf{h}_T, \tag{1}$$

where $\Delta\mathbf{W}_{i,T}^{(Q)} = \alpha_{i,T}\mathbf{A}_{i,T}\mathbf{B}_{i,T}$ is the low-rank update introduced by L_i as defined in Section 3.1. From each projection, we compute a scalar *signal score* s_i that quantifies the relevance of adapter L_i to the current input. Typical examples include the ℓ_2 norm,

$$s_i = \|\mathbf{o}_{i,T}\|_2, \tag{2}$$

or entropy-based measures,

$$p_{i,T}^{(j)} = \frac{\exp(o_{i,T}^{(j)})}{\sum_k \exp(o_{i,T}^{(k)})},$$
$$s_i = \left(-\sum_j p_{i,T}^{(j)} \log p_{i,T}^{(j)}\right)^{-1}, \tag{3}$$

derived from the projection distribution, both of which capture how strongly a LoRA adapter responds to the input. Intuitively, a larger projection norm indicates stronger activation and thus greater influence on the model's output, while lower-entropy projections imply more confident and focused responses. Hence, these metrics serve as natural indicators of adapter relevance. Here, we take the reciprocal of entropy so that scores are always positive and lower-entropy (i.e., more confident) adapters receive larger weights.

To examine whether these projection-based signals indeed capture task relevance, Fig. 2 demonstrates signal intensities across existing LoRA adapters (from the pool) and datasets. Each column represents a LoRA trained on a specific dataset, while each row represents the dataset used for inference. The heatmap values indicate the normalized signal (here, norm) when each LoRA is applied to samples from different datasets. A clear block-diagonal pattern emerges, highlighted with red boxes, revealing that ***similar tasks activate LoRAs in similar ways***. This observation provides empirical evidence that the extracted signals reflect meaningful semantic relationships among tasks and can effectively guide adapter selection without any additional training.

Finally, based on the collection of signals $\{s_i\}_{i=1}^N$, we select the *top-k adapter set*

$$\mathcal{S} = \mathrm{TopK}\left(\{(L_i, s_i)\}_{i=1}^N, k\right), \tag{4}$$

which contains the k adapters with the highest scores. This set $\mathcal{S}$ serves as the candidate pool for merging in the next stage (Section 3.3).

3.3 Merging LoRAs

After selecting the top-k adapters $\mathcal{S}$ as described in Section 3.2, the next step is to merge them into the base model. Following prior work (Zhao et al., 2024), we consider two types of merging strategies:

- **Output-based Merging (Mixture).** The projection outputs $\{\mathbf{p}_{i,T}\}_{i\in\mathcal{S}}$ are combined directly at the output level. That is, given hidden input $\mathbf{h}_T$, the merged projection is formed as a weighted sum of the selected adapters' projections.
- **Parameter-based Merging (Fusion).** The low-rank parameter updates $\{\Delta\mathbf{W}_{i,T}^{(Q)}\}_{i\in\mathcal{S}}$ are merged into a single fused update, which is then re-attached to the base model.

While both strategies are possible, LoGo adopts *output-based merging* for efficiency. Parameter-based fusion requires explicit recomputation of the merged weight matrices and re-attaching into the model at every step, which introduces significant overhead in deployment scenarios with many adapters. In contrast, output-based merging avoids additional overhead of parameter-level operations, since it directly discards unselected LoRAs during token generation and combines only the projections of the selected ones.

Formally, let s_i be the signal score of adapter $L_i \in \mathcal{S}$. We normalize these scores into non-negative weights

$$\tilde{w}_i = \frac{s_i}{\sum_{j\in\mathcal{S}} s_j}, \quad i \in \mathcal{S}. \tag{5}$$

The merged projection is then given by

$$\mathbf{o}_{\text{merge}} = \sum_{i\in\mathcal{S}} \tilde{w}_i\, \mathbf{o}_{i,T}. \tag{6}$$

In practice, this weighted summation can be efficiently implemented by adjusting only the scaling factors of the selected adapters, without modifying

or reloading their parameters. This design enables LoGo to adaptively merge multiple LoRAs with minimal runtime overhead, while maintaining flexibility to handle diverse inputs on the fly.

4 Experiments

We conduct extensive experiments to evaluate the performance and computational efficiency of LoGo. Section 4.1 outlines the experimental setup, including the base models, datasets, and baselines. Section 4.2 reports the performance of LoGo across diverse datasets and in mixed-dataset scenarios, followed by Section 4.3, which analyzes the inference-time throughput of our method.

4.1 Evaluation Setup

Base Models and LoRA Adapters. We use **LLaMA-3.1-8B**, **Qwen-2.5-7B**, and **DeepSeek-LLM-7B-Base** as the base pretrained models in our evaluation. For each model, we train **260 LoRA adapters** on distinct Flan-v2 tasks (Wei et al., 2022; Chung et al., 2024), and then evaluate LoGo as well as other baseline methods using the corresponding pretrained model with these adapters. More details on LoRA training are in Appendix C.1. **We plan to publicly release the adapters trained on all pretrained models.**

Datasets. We evaluate LoGo on a diverse set of benchmarks spanning multiple task categories. For **BIG-Bench Hard (BBH)** (Suzgun et al., 2023), we include Boolean Expressions, Causal Judgement, Formal Fallacies, Navigate, Object Counting, Sports Understanding, Web of Lies, and Word Sorting. For **Machine Translation**, we use datasets from the WMT benchmarks (Bojar et al., 2014, 2016), including WMT'14 FR→EN, WMT'14 EN→FR, WMT'16 DE→EN, WMT'16 EN→DE, WMT'16 RO→EN, and WMT'16 EN→RO. For **Struct-to-Text Generation**, we adopt datasets from the GEM benchmark (Gehrmann et al., 2021), including CommonGen (Lin et al., 2020), DART (Nan et al., 2021), E2ENLG (Novikova et al., 2017), and WebNLG (Gardent et al., 2017). For **Closed Book Question Answering**, we use ARC-c, ARC-e (Clark et al., 2018), Natural Questions (Kwiatkowski et al., 2019), and TriviaQA (Joshi et al., 2017). Finally, for **Natural Language Inference**, we evaluate on ANLI-R1, ANLI-R2, ANLI-R3 (Nie et al., 2020), and QNLI (Wang et al., 2018). This collection covers reasoning, translation, structured generation, question answering, and inference tasks, providing a comprehensive evaluation of LoGo under diverse conditions.

Baselines. We compare LoGo against three baselines: **Base**, the base pretrained model without any LoRA adapters; **LoRAHub** (Huang et al., 2024), which learns weights to merge the parameters of LoRA adapters via weighted summation; and **LoRARetriever** (Zhao et al., 2024), which trains an auxiliary language model to retrieve the most relevant adapters for a given input based on embedding similarity. For LoRARetriever, we report results with both *mixture* and *fusion* merging, as suggested in the original paper. We use the implementation of all baseline methods following their guidelines.

Implementation Details. For all baselines, we fix the number of selected and merged LoRA adapters to 20, while keeping all other hyperparameters consistent with the default settings of the respective pretrained models used in our experiments. We use the last Transformer block of each model as the target block for signal extraction, denoted as B_T. Similarly, the signal is extracted from the last token of each input sequence. Ablation studies on the number of selected adapters, the choice of Transformer block, and the token used for signal extraction are presented in Appendix D.1, which show that LoGo maintains stable performance across these configurations.

4.2 Evaluation Results

Main Results Table 1 reports the performance of LoGo with norm- and entropy-based signals compared to baseline methods on LLaMA-3.1-8B, Qwen-2.5-7B, and DeepSeek-LLM-7B-Base across multiple datasets. LoGo consistently outperforms the baselines in many tasks and remains competitive in the rest. This outcome is remarkable given that LoGo requires no additional training, whereas the baseline methods rely on fine-tuning.

Mixed-dataset Scenario To further assess the generalization ability of LoGo beyond the training domains of the LoRA adapters, we evaluate it on **CodeXGLUE** (Lu et al., 2021), a benchmark comprising diverse programming-language tasks unseen during adapter training. Specifically, we consider five subtasks: Code Refinement, Code Translation (Java→C#), Code Translation (C#→Java), Code-to-Text (Java), and Code-to-Text (Python). All tasks are evaluated using the BLEU metric.

As shown in Table 2, LoGo outperforms all

		Llama-3.1-8B						Qwen-2.5-7B						Deepseek-LLM-7B-Base					
Task	Metric	Base	LoRA-Hub	LoRA-Retriever (Mixture)	LoRA-Retriever (Fusion)	LoGo (Norm)	LoGo (Entropy)	Base	LoRA-Hub	LoRA-Retriever (Mixture)	LoRA-Retriever (Fusion)	LoGo (Norm)	LoGo (Entropy)	Base	LoRA-Hub	LoRA-Retriever (Mixture)	LoRA-Retriever (Fusion)	LoGo (Norm)	LoGo (Entropy)
BBH																			
Boolean Expressions	EM	58.0	64.5	**76.7**	67.3	71.3	**76.7**	76.7	76.7	**77.3**	**77.3**	74.0	74.7	66.0	55.9	66.7	**68.0**	**68.0**	66.0
Causal Judgement	EM	41.4	**54.0**	**54.0**	50.6	48.3	47.1	59.8	51.5	59.8	59.8	**62.1**	59.8	27.6	43.4	**48.3**	**48.3**	**48.3**	14.9
Formal Fallacies	EM	25.3	42.7	49.3	50.7	**52.0**	50.0	53.3	53.2	**56.7**	**56.7**	54.7	53.3	0.7	31.7	**48.7**	**48.7**	**48.7**	0.0
Navigate	EM	45.3	48.5	**56.7**	46.0	50.0	50.0	50.7	**52.0**	50.7	50.7	50.7	50.7	46.0	51.7	**54.0**	49.3	51.3	53.3
Object Counting	EM	35.3	**39.1**	20.7	37.3	27.3	24.0	41.3	35.2	40.7	40.7	**44.0**	43.3	46.0	40.9	44.0	**46.7**	44.0	38.7
Sports Understanding	EM	0.0	**8.9**	0.0	0.0	1.3	3.3	22.7	73.1	76.7	76.7	**80.0**	78.7	0.0	**26.8**	0.0	0.0	0.0	0.7
Web-of-lies	EM	2.0	21.6	**32.0**	22.0	14.7	26.7	5.3	41.6	54.0	**55.3**	49.3	51.3	0.0	**13.9**	0.0	0.0	0.0	0.0
Word Sorting	EM	11.3	16.7	34.0	29.3	41.3	**42.0**	0.0	1.9	**12.7**	12.0	12.0	9.3	3.3	4.4	**7.3**	**7.3**	5.3	1.3
Average		27.3	37.0	**40.4**	37.9	38.3	40.0	38.7	48.1	**53.6**	**53.6**	53.3	52.6	23.7	**33.6**	**33.6**	33.5	33.2	21.9
Translation																			
WMT'14 FR→EN	BLEU	27.7	27.1	**28.7**	28.5	28.6	28.6	28.5	6.0	30.3	30.2	**30.4**	30.3	26.0	25.4	**27.2**	27.0	27.0	26.5
WMT'14 EN→FR	BLEU	25.3	25.2	**27.7**	27.5	27.2	27.5	27.8	5.7	**28.8**	28.7	28.7	28.7	19.3	21.3	22.2	22.0	**22.5**	22.0
WMT'16 DE→EN	BLEU	29.8	29.8	30.7	30.1	**31.0**	30.9	28.8	9.5	**31.7**	31.3	31.5	31.4	27.2	27.0	28.1	27.9	**28.6**	27.8
WMT'16 EN→DE	BLEU	20.0	19.4	21.3	20.9	21.6	**21.8**	20.0	4.2	**20.6**	**20.6**	**20.6**	**20.6**	12.6	13.9	14.5	14.4	**15.0**	14.3
WMT'16 RO→EN	BLEU	27.9	23.9	**29.1**	28.6	28.1	28.7	25.5	6.4	**29.1**	29.0	28.9	28.9	25.9	19.5	**27.1**	26.8	26.7	26.0
WMT'16 EN→RO	BLEU	16.3	16.1	18.3	18.1	17.7	**18.4**	15.5	1.1	15.5	15.5	**15.7**	**15.7**	11.0	12.5	**14.2**	14.0	14.1	13.5
Average		24.5	23.6	25.9	25.6	25.7	**26.0**	24.4	5.5	**26.0**	25.9	25.9	25.9	20.3	19.9	22.2	22.0	**22.3**	21.7
Struct-to-Text																			
CommonGen	Rouge-1	54.8	46.8	55.5	54.3	**56.0**	**56.0**	52.3	33.7	**54.7**	54.4	**54.7**	54.3	0.0	51.9	50.8	49.8	52.5	**53.5**
	Rouge-2	23.6	20.4	24.9	24.1	24.7	**25.0**	21.5	14.0	**23.1**	22.8	**23.1**	22.5	0.0	20.9	21.5	20.4	21.8	**22.7**
	Rouge-L	44.0	38.1	45.4	44.0	45.5	**45.8**	41.6	27.6	44.7	44.5	**44.8**	44.4	0.0	42.7	42.8	41.8	44.0	**44.7**
DART	Rouge-1	63.9	53.8	63.6	64.1	67.6	**68.7**	71.0	4.9	73.0	73.0	73.2	**73.3**	0.5	61.3	60.4	60.0	**66.5**	50.6
	Rouge-2	37.9	32.2	37.6	37.8	40.9	**42.0**	44.2	1.8	47.1	47.2	47.4	**47.5**	0.3	36.4	35.4	35.1	**40.0**	27.9
	Rouge-L	47.9	40.7	48.1	48.3	50.6	**51.4**	53.8	4.5	55.9	**56.1**	**56.1**	56.0	0.4	47.3	46.8	46.7	**49.4**	38.1
E2ENLG	Rouge-1	65.4	53.6	64.7	64.6	**69.0**	**69.0**	70.0	24.8	70.9	70.9	**71.0**	**71.0**	0.0	58.6	57.8	56.8	**65.3**	60.7
	Rouge-2	35.8	30.0	35.6	35.3	38.8	**39.0**	41.1	13.8	42.5	42.5	**42.7**	42.6	0.0	31.9	30.6	30.0	**36.2**	32.2
	Rouge-L	44.6	36.6	45.1	44.6	**46.9**	46.8	48.6	18.6	49.7	49.7	**49.8**	49.6	0.0	42.1	41.6	41.1	**44.9**	42.3
WebNLG	Rouge-1	58.4	64.2	63.7	64.2	68.1	**69.0**	72.3	18.3	72.6	**72.7**	72.6	72.6	0.0	54.4	60.8	61.0	**65.7**	55.5
	Rouge-2	35.8	38.9	38.2	38.5	41.8	**42.5**	46.4	10.8	**47.1**	**47.1**	46.5	46.4	0.0	33.2	37.8	38.1	**40.7**	32.7
	Rouge-L	45.1	49.5	49.1	49.2	52.1	**53.1**	55.8	14.7	**56.6**	**56.6**	56.3	56.3	0.0	44.0	48.8	49.1	**51.2**	43.7
Average		46.4	42.1	47.6	47.4	50.2	**50.7**	51.6	15.6	**53.2**	53.1	**53.2**	53.0	0.1	43.7	44.6	44.2	**48.2**	42.0
Closed Book QA																			
ARC-c	EM	64.7	64.8	67.9	67.7	69.1	**69.9**	85.9	86.3	**86.9**	**86.9**	86.6	**86.9**	0.0	45.1	45.1	**49.1**	33.0	28.2
ARC-e	EM	78.7	69.4	79.0	79.1	**81.3**	81.2	**91.6**	22.3	**91.6**	91.4	**91.6**	**91.6**	0.0	58.0	56.7	**63.4**	43.6	37.2
Natural Questions	EM	5.3	7.1	12.6	**13.1**	11.1	11.7	4.8	6.8	**12.8**	12.7	12.4	12.3	8.7	8.9	10.2	**10.5**	9.0	9.1
Trivia QA	EM	12.7	12.2	**15.0**	**15.0**	14.6	14.5	6.6	9.1	**11.6**	**11.6**	10.5	10.8	10.9	11.4	**12.1**	**12.1**	11.8	11.3
Average		40.4	38.4	43.6	43.7	44.0	**44.3**	47.2	31.1	**50.7**	**50.7**	50.3	50.4	4.9	30.8	31.0	**33.8**	24.4	21.4
Natural Language Inference (NLI)																			
ANLI-R1	EM	35.0	35.2	38.4	38.4	42.0	**42.5**	55.5	58.6	61.2	60.6	61.3	**61.5**	7.5	12.8	33.5	33.3	**33.6**	33.2
ANLI-R2	EM	32.6	31.4	40.5	38.9	42.4	**42.8**	45.9	48.9	53.3	52.0	53.2	**53.7**	8.6	30.2	33.6	**33.9**	33.4	33.1
ANLI-R3	EM	37.3	23.1	41.9	42.7	43.0	**43.4**	51.2	52.4	54.2	53.7	**54.3**	54.2	4.0	16.3	33.4	**33.8**	33.5	33.2
QNLI	EM	31.0	**41.2**	7.5	10.3	16.3	19.9	85.8	46.2	86.1	**86.2**	86.1	86.0	0.2	**4.3**	2.4	2.7	1.4	4.0
Average		34.0	32.7	32.1	32.6	35.9	**37.2**	59.6	51.5	63.7	63.1	63.7	**63.8**	5.1	15.9	25.7	**25.9**	25.5	**25.9**

Table 1: Performance of LoGo (with norm- and entropy-based signals) compared to baselines across diverse tasks on the LLaMA-3.1-8B, Qwen-2.5-7B, and DeepSeek-LLM-7B-Base backbones. The best results are in **bold**, the second-best are underlined, and our results are highlighted with a blue background. For Struct-to-Text tasks, the reported average is computed over all ROUGE metrics (Rouge-1, Rouge-2, and Rouge-L).

	Lora-Hub	LoRA-Retriever (Mixture)	LoRA-Retriever (Fusion)	LoGo (Norm)	LoGo (Entropy)
Code Refinement	33.5	29.2	42.1	**46.3**	41.5
Code Translation (Java → C#)	9.7	10.7	6.3	**11.2**	10.6
Code Translation: (C# → Java)	10.4	9.4	**13.0**	7.6	11.7
Code-to-Text (Java)	**1.5**	1.2	1.0	1.0	1.1
Code-to-Text (Python)	3.2	1.5	4.2	**6.1**	3.7
Average	11.6	10.4	13.3	**14.4**	13.7

Table 2: Comparison of LoGo with baselines on unseen mixed-dataset scenarios using the LLaMA-3.1-8B backbone, evaluated on the CodeXGLUE dataset. The best results are in **bold**, the second-best are underlined, and our results are highlighted with a blue background.

	Training	Inference					
	LoRA-Hub	Base	LoRA-Hub	LoRA-Retriever (Mixture)	LoRA-Retriever (Fusion)	Ours (Norm)	Ours (Entropy)
Boolean Expressions	24.52	0.37	1.76	1.83	2.44	2.87	1.73
Causal Judgement	47.59	0.32	0.36	1.91	1.97	1.98	2.07
Formal Fallacies	25.29	0.44	0.39	1.85	1.88	1.77	1.65
Navigate	21.87	0.40	0.92	1.89	1.88	1.86	1.97
Object Counting	23.63	0.41	1.57	2.06	2.08	2.36	1.75
Sports Understanding	16.20	0.72	0.73	1.99	2.02	1.86	1.67
Web-of-lies	17.61	0.38	1.85	2.38	2.93	1.62	1.93
Word Sorting	17.51	0.76	1.61	2.29	2.36	2.32	2.22
Average	24.28	0.47	1.15	2.03	2.19	2.08	1.87

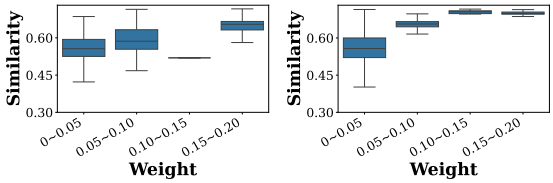
Table 3: Per-sample inference time (in seconds) of LoGo compared with baseline methods for Llama-3.1-8B model. For LoRAHub, we additionally report training time required to learn task-specific merging weights.

baselines on average. These results indicate that the signal-based selection and merging mechanism in LoGo generalizes effectively to unseen domains, capturing cross-task relevance even when input distributions differ substantially from those of the adapters' training data.

4.3 Computation Time

We also analyze the computational cost of LoGo relative to baseline methods. Table 3 reports per-sample inference times on the LLaMA-3.1-8B model using a single NVIDIA H100 GPU. As expected, the base pretrained model is the fastest, since no adapters are attached. LoRAHub introduces additional overhead and requires training to learn task-specific merging weights (24.28 seconds on average), which limits its practicality when new adapters are frequently introduced.

LoGo and LoRARetriever exhibit comparable inference times ($\sim$ 2 sec/sample), reflecting the cost of adapter-level operations. Since Lo-



(a) Norm (b) Entropy

Figure 3: Alignment between merging weights and task similarity of LoGo with (a) norm and (b) entropy as signals for Big-Bench Hard task and Qwen2.5-32B model.

RARetriever depends on maintaining task datasets and training an auxiliary embedding model, these results underscore the practicality of LoGo.

5 Analysis

We conduct a series of analyses to gain deeper insights into the behavior and design choices of LoGo. Section 5.1 examines the alignment between merging weights and task similarity, Section 5.2 analyzes the characteristics of selected LoRAs, Section 5.3 compares different merging strategies, and Section 5.4 highlights the benefits of LoGo in long-generation scenarios.

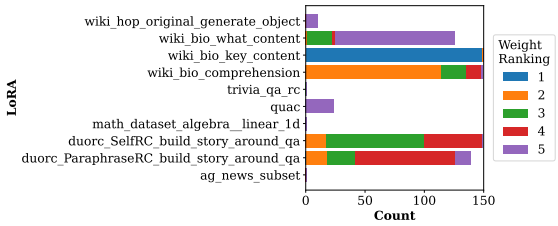
5.1 Alignment with Task Similarity

We examine whether the merging weights produced by LoGo align with task similarity. Since weights in LoGo are derived from projection-based signals, LoRAs trained on similar tasks should assign higher weights to each other's inputs.

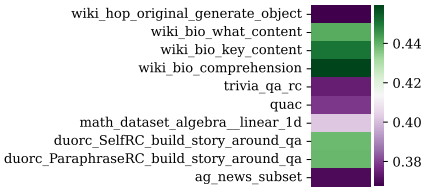
Task similarity is measured as the average cosine similarity, computed from the pretrained model's embeddings between the given input and samples from the FLAN-v2 dataset. Fig. 3 shows a box plot with merging weight on the x-axis and task similarity on the y-axis. The results display a clear upward trend: LoRAs given larger weights correspond to more similar tasks. An exception occurs in the norm-based setting in the weight bucket 0.10–0.15, which appears anomalous because it arises from a single instance out of 22,740 cases (1137 samples $\times$ 20 selected LoRAs). This isolated case manifests as an outlier in the distribution. This confirms that LoGo's signal-based weighting not only enables efficient merging but also captures semantic relations between tasks without additional training.

5.2 Analysis of Selected LoRAs

In this section, we analyze the LoRA adapters selected by LoGo. We employ LoGo with the LLaMA-3.1-8B model on the BBH Word Sorting task and report the selection counts of LoRA adapters in Figure 4a. Each color within a bar represents the ranking of a LoRA adapter when it was selected. Results show that certain LoRA adapters are consistently selected across samples, although their relative priorities vary considerably. We fur-



(a) Lora Selection Count



(b) Task Similarity

Figure 4: Comparison of (a) LoRA selection count by LoGo with Llama-3.1-8B model and (b) task similarity for BBH Word Sorting dataset. Each color in the bar present the priority of the LoRA when it was selected.

	Sample A	Sample B
Question	Sort the following words alphabetically: List: thunderclap swab built poland	Sort the following words alphabetically: List: sanhedrin scratchy helical beau venezuela awash bessie extricable indoeuropean vice pendulum cream animism
Answer	built poland swab thunderclap	animism awash beau bessie cream extricable helical indoeuropean pendulum sanhedrin scratchy venezuela vice

(a) Samples from BBH Word Sorting

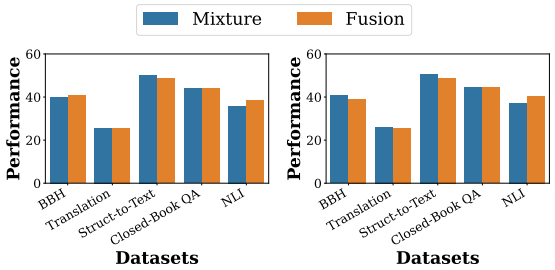
	Sample A	Sample B
Question	Given a meaning representation, write a short and simple sentence that contains all the information in the meaning representation. Meaning Representation: [eatType[pub], food[Fast food], customer rating[high], area[riverside], familyFriendly[no], near[Café Rouge]]	Given a meaning representation, write a short and simple sentence that contains all the information in the meaning representation. Meaning Representation: [name[Blue Spice], eatType[pub], food[Chinese], area[city centre], near[Rainbow Vegetarian Café]]
Answer	The Mills is not kid friendly as it is a riverside pub near Café Rouge. Its mid priced fast food is highly rated.	Blue Spice, located near Rainbow Vegetarian Café in the city centre, is a pub that also sells Chinese food. Children should not visit.

(b) Samples from E2ENLG

Dataset	Common	Only in A	Only in B
BBH Word Sorting	wiki_bio_key_content, wiki_bio_what_content	ag_news_subset, math_dataset_algebra __linear_1d, trivia_qa_rc	duorc_SelfRC_build_ story_around_qa, wiki_bio_comprehension, wiki_hop_original_ generate_object
E2ENLG	duorc_ParaphraseRC_ build_story_around_qa, duorc_SelfRC_build_ story_around_qa, wiki_bio_comprehension, wiki_bio_key_content, wiki_bio_what_content	N/A	N/A

(c) LoRA adapters selected by LoGo

Table 4: Comparison of top-5 selected LoRA adapters for two samples from the BBH Word Sorting and E2ENLG dataset. For the E2ENLG dataset, selected LoRA adapters are common across both samples.



(a) Norm (b) Entropy

Figure 5: Performance of LoGo using (a) norm and (b) entropy across datasets with different merging methods – mixture and fusion.

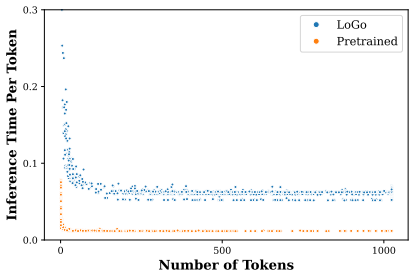


Figure 6: Inference time per token with varying numbers of tokens in CNN-DailyMail dataset samples.

ther compare these selection frequencies with each adapter's task similarity to the BBH Word Sorting dataset, as shown in Figure 4b. This confirms that LoRAs trained on tasks more similar to the target dataset tend to be selected more frequently.

Anecdotal Examples: We compare the LoRA selections of LoGo for two samples from the BBH Word Sorting and E2ENLG dataset in Table 4. For BBH Word Sorting, a small set of general-purpose adapters (e.g., wiki_bio family) are consistently activated across both samples. These modules are typically associated with reasoning and factual comprehension, providing a general foundation across tasks. Beyond these common adapters, Sample A relies on adapters related to summarization (e.g., ag_news), which facilitate short and structured outputs. In contrast, Sample B engages modules emphasizing long story generation ability (e.g., duorc family). Meanwhile, E2ENLG commonly selects general-purpose comprehension modules, as its story-generation task involves casual content that does not require domain-specific expertise.

5.3 Analysis on Merging Method

LoGo adopts the mixture as merging strategy, where the projection outputs of selected adapters are combined. To assess this choice, we conduct an ablation study by replacing mixture with fusion, which merges the parameters of the selected adapters into a single set of weights before reattaching them to the model. Results are summarized in Fig. 5, which compares the two strategies on the Llama-3.1-8B model across five task categories: BBH, Translation, Struct-to-Text, Closed-Book QA, and NLI. The reported results are averaged over each dataset category.

Overall, we find that mixture and fusion achieve comparable performance across all categories. However, fusion requires heavy parameter recomputation and re-attaching at the instance level, which introduces a substantial computational burden in deployment scenarios with large adapter pools. In contrast, mixture achieves similar accuracy while incurring much lower computational overhead, making it a more practical choice.

5.4 Benefits in Long Generation Task

We analyze how the computational overhead of LoGo behaves in long text generations. Experiments are conducted on the **CNN-DailyMail** (See et al., 2017) dataset using the LLaMA-3.1-8B model with an NVIDIA H200 GPU. We measure the inference time per token as a function of the number of generated tokens, and compare LoGo against the pretrained model.

As shown in Figure 6, the per-token inference time of LoGo decreases rapidly as the number of generated tokens increases, stabilizing after approximately 100 tokens. While a gap remains between LoGo and the pretrained model, this difference reflects the intrinsic overhead of employing LoRA adapters. These results indicate that the cost of signal extraction at the beginning of generation is effectively amortized over longer sequences.

6 Conclusion

We present LoRA on the Go (LoGo), a training-free framework that dynamically selects and merges LoRA adapters in deployment scenarios without any task-specific training. By extracting lightweight relevance signals from a single forward pass, LoGo identifies the most suitable adapters for each input and merges them through signal-weighted summation. Experiments across diverse NLP benchmarks show that LoGo achieves comparable or better performance than training-based baselines, while maintaining inference throughput. These results highlight the potential of training-free, instance-specific adaptation as a promising direction for deploying large language models in real-world, heterogeneous environments.

Limitations

While LoGo demonstrates strong performance without any task-specific training, several limitations remain. First, our approach relies on projection-based signals extracted from a single forward pass. Although effective in practice, this mechanism does not guarantee that the selected adapters always align with task relevance, particularly in highly out-of-distribution scenarios. Second, our experiments primarily use adapters fine-tuned on the FLAN-v2 dataset. Extending evaluation to LoRAs trained on diverse domains (e.g., multimodal, or low-resource data) would help assess generality. Third, the current framework assumes access to a shared pretrained model and a large pool of LoRA adapters. Attaching many adapters simultaneously can increase memory usage and slow inference, making optimization of adapter management (e.g., pruning or selective loading) an important direction for future work.

References

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, and 1 others. 2024. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*.

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, and 1 others. 2014. Findings of the 2014 workshop on statistical machine translation.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurélie Névéol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, and 2 others. 2016. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 632–642. Association for Computational Linguistics.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, and 1 others. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Wenfeng Feng, Chuzhan Hao, Yuewei Zhang, Yu Han, and Hao Wang. 2024. Mixture-of-loras: An efficient multitask tuning for large language models. *COLING*.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. Creating training corpora for nlg micro-planning.

Sebastian Gehrmann, Tosin Adewumi, Karmanya Aggarwal, Pawan Sasanka Ammanamanchi, Anuoluwapo Aremu, Antoine Bosselut, Khyathi Raghavi Chandu, Miruna Clinciu, Dipanjan Das, Kaustubh Dhole, and 1 others. 2021. The gem benchmark: Natural language generation, its evaluation and metrics.

Google Gemini. 2023. Gemini. https://gemini.google.com/.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. *ICML*.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. 2024. Lorahub: Efficient cross-task generalization via dynamic lora composition. *ICLR*.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, and 1 others. 2019. Natural questions: A benchmark for question answering research.

Hector Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. In *Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

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

Bill Yuchen Lin, Wangchunshu Zhou, Ming Shen, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. 2020. Commongen: A constrained text generation challenge for generative commonsense reasoning.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *NeurIPS*.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, and 1 others. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. PEFT: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft.

Microsoft. 2023. Copilot. https://www.microsoft.com/en-us/copilot.

Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, and 1 others. 2021. Dart: Open-domain structured data record to text generation.

Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2020. Adversarial nli: A new benchmark for natural language understanding.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The e2e dataset: New challenges for end-to-end generation.

OpenAI. 2023. Chatgpt. https://chat.openai.com.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, and 2 others. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, and 1 others. 2023. Challenging big-bench tasks and whether chain-of-thought can solve them.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2022. Finetuned language models are zero-shot learners. *ICLR*.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1112–1122. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clément Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*. Also available at https://huggingface.co.

Xun Wu, Shaohan Huang, and Furu Wei. 2024. Mixture of lora experts. *ICLR*.

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

A Signals from LoRA Projections

We analyze the signal intensity of LoRA projections across datasets for the Llama-3.1-8B and Qwen-2.5-7B models, using both norm and entropy as metrics. Fig. 7 presents a heatmap where columns correspond to LoRAs trained on different tasks and rows correspond to datasets from those tasks. Clear block structures emerge, with related tasks highlighted by red boxes, suggesting that similar tasks activate LoRAs in similar ways.

B The LoGo Algorithm

Algorithm 1 summarizes the overall procedure of LoGo.

Algorithm 1 LoGo: Dynamic Selection and Merging of LoRA Adapters

Input: Hidden input $\mathbf{h}_T$ at target block B_T, N tasks LoRAs $\mathcal{L} = \{L_i\}_{i=1}^{N}$, the respective low rank Q-projection update $\Delta W_{i,T}^{(Q)}$, top-k parameter k, scoring method
Output: Merged adapter projection $\mathbf{o}_{\text{merge}}$
Probe pass: Compute each LoRA's projection output
for $i \leftarrow 1 \ldots N$ **do**
  $\mathbf{o}_{i,T} \leftarrow \Delta \mathbf{W}_{i,T}^{(Q)} \mathbf{h}_T$
  if scoring method = ℓ_2 **then**
    $s_i \leftarrow \|\mathbf{o}_{i,T}\|_2$
  else if scoring method = entropy **then**
    $p \leftarrow \text{softmax}(\mathbf{o}_{i,T})$
    $s_i \leftarrow 1/\Big(-\sum_j p_j \log p_j\Big)$
  end if
end for
Adapter selection:
$\mathcal{S} \leftarrow \text{TopK}(\{(L_i, s_i)\}, k)$
Normalize weights $\tilde{w}_i = s_i / \sum_{j \in \mathcal{S}} s_j$ for each $L_i \in \mathcal{S}$
Output-based merging:
$\mathbf{o}_{\text{merge}} \leftarrow \sum_{i \in \mathcal{S}} \tilde{w}_i \mathbf{o}_{i,T}$
return $\mathbf{o}_{\text{merge}}$

C Additional Details on Experiments

C.1 Details on LoRA Training

We split each Flan-v2 dataset into training, validation, and test sets with an 8:1:1 ratio. We trained the LoRA adapter with a per-device batch size of 4 and gradient accumulation of 16, resulting in an effective batch size of 64. The learning rate was set to 2×10^{-4}, and training was conducted for 20 epochs. The best model checkpoint was selected based on validation loss. A full list of the Flan-v2 datasets used for training is provided in Table 5.

C.2 Implementation Details

We implement LoGo based on Pytorch (Paszke et al., 2019), Huggingface (Wolf et al., 2020), and PeFT library (Mangrulkar et al., 2022). Specifically we utilize *PeftMixedModel* class, which allows multiple adapters simultaneously and control their scales. We fix the number of selected and merged LoRA adapters to 20, while keeping all other hyperparameters consistent with the default settings of the respective pretrained models used in our experiments. We use the last Transformer block of each model as the target block for signal extraction. Similarly, the signal is extracted from the last token of each input sequence. Since the LoRA adapters are fixed and their respective signals (norm and entropy) are deterministic, we conduct our experiment over a single run.

C.3 Details on Evaluation Datasets

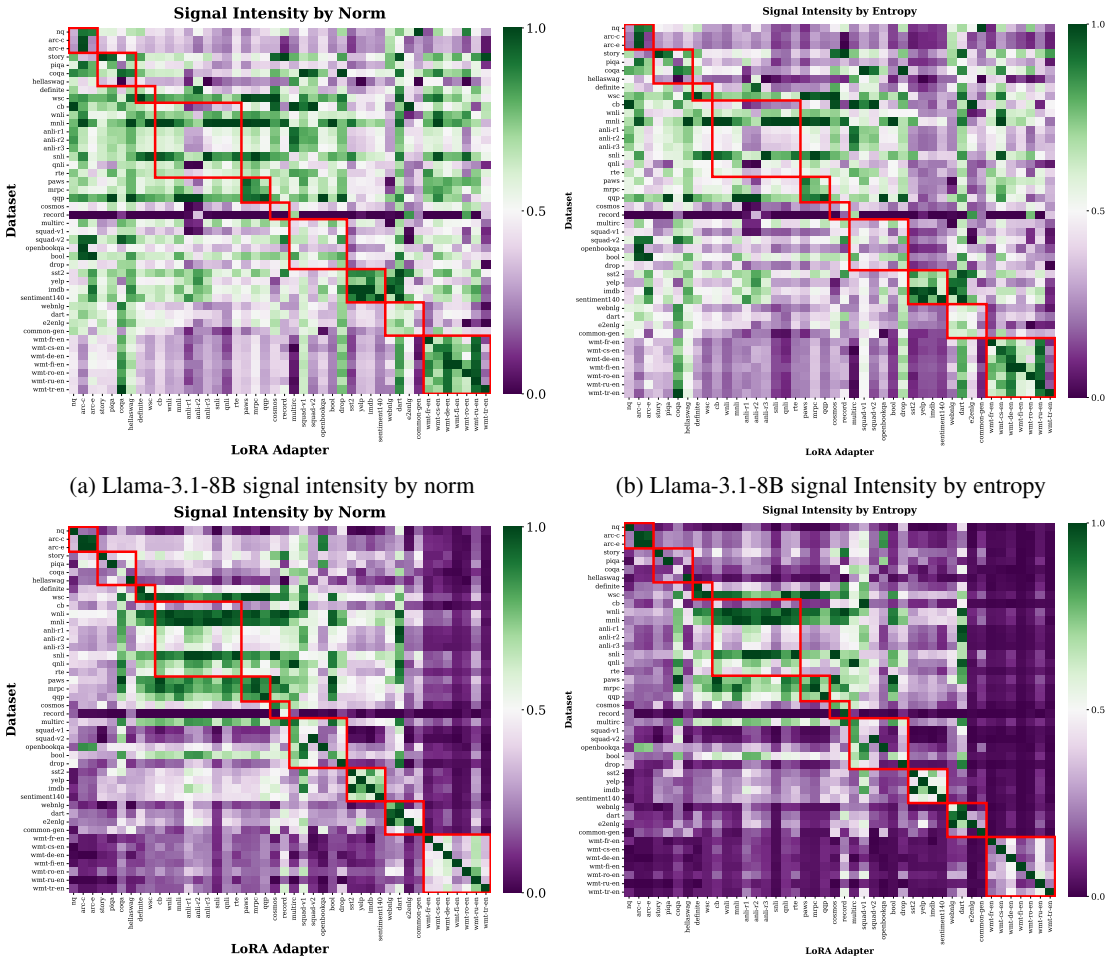
In our evaluation, we used the designated test split of each dataset. When test labels were not available, we relied on the validation or development split for evaluation. For training LoRAHub, we sampled five instances from the training split. For the BIG-Bench Hard datasets, we followed the same train–test split configuration as LoRAHub.

D Additional Results

D.1 Ablation Studies

In this section, we conduct ablation studies on LoGo to examine the effects of the token used for signal extraction, the number of selected modules, and the specific block used for signal extraction. All experiments are performed using LoGo with Llama-3.1-8B on the BIG-Bench Hard, Translation, Struct-to-Text, Closed-Book QA, and Natural Language Inference tasks. Evaluation metrics are BLEU for translation, ROUGE for Struct-to-Text, and Exact Match for the other datasets. The reported results are averaged over each dataset category.

Token for Signal Extraction In LoGo, signals are extracted from the projection outputs corresponding to the last token of the input, which serves as our default setting. To investigate the effect of this choice, we compare three alternatives: using the first token, the last token, and the average across all tokens. Results are shown in Fig. 8, which presents bar plots comparing all alternatives. Across both norm- and entropy-based scoring, we observe that the performance differences between token choices are small, indicating that LoGo is robust to this design decision. Among the three options, however, the last token consistently achieves



(a) Llama-3.1-8B signal intensity by norm

(b) Llama-3.1-8B signal Intensity by entropy

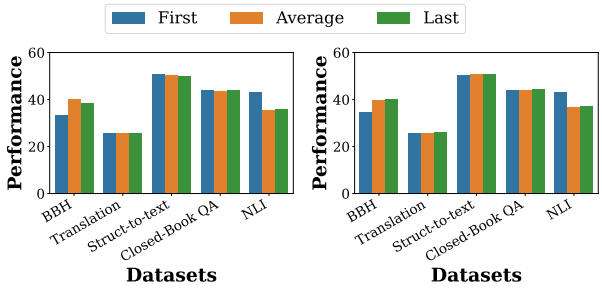
(c) Qwen-2.5-7B signal intensity by norm

(d) Qwen-2.5-7B signal Intensity by entropy

Figure 7: Heatmaps illustrating signal intensity patterns across LoRA adapters trained on top of the LLaMA-3.1-8B and Qwen-2.5-7B backbone. The x-axis represents LoRAs trained on different tasks, while the y-axis corresponds to datasets from those tasks. Each cell shows the norm or (1 / entropy) of the projection outputs. The cell values are min-max normalized across datasets for each LoRA. The related tasks are highlighted in red boxes.

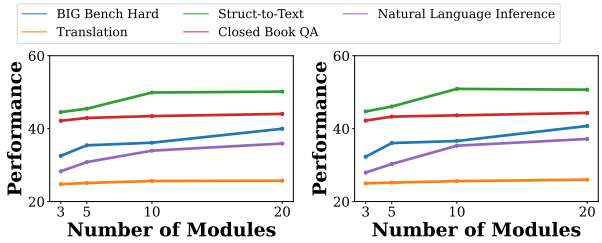
Category	Datasets (Flan-v2 subsets used)
Question Answering	adversarial_qa_dbert_*, adversarial_qa_dbidaf_*, adversarial_qa_droberta_*, ai2_arc_ARC-Challenge, ai2_arc_ARC-Easy, bool_q, coqa, cosmos_qa, drop, duorc_ParaphraseRC_*, duorc_SelfRC_*, hotpotqa (kilt_tasks_hotpotqa_*), natural_questions_open, openbookqa, qasc_*, quac, quail_*, quarel_*, quartz_*, quoref_*, race_high_*, race_middle_*, ropes_*, sciq_*, squad_v1.1, squad_v2.0, trivia_qa_rc, unified_qa_science_inst, web_questions_*, wiki_hop_original_*, wiki_qa_*
Natural Language Inference	anli_r1, anli_r2, anli_r3, glue_mnli, glue_rte, glue_wnli, snli, super_glue_cb, super_glue_copa, super_glue_multirc, super_glue_record, super_glue_wic, super_glue_wsc.fixed
Classification / Sentiment	ag_news_subset, amazon_polarity_*, app_reviews_*, dbpedia_14_*, glue_cola, glue_mrpc, glue_qqp, glue_sst2, imdb_reviews_plain_text, opinion_abstracts_idebate, opinion_abstracts_rotten_tomatoes, sentiment140, trec, yelp_polarity_reviews
Commonsense Reasoning	cos_e_v1.11_*, hellaswag, lambada, piqa, social_i_qa_*, story_cloze_2016, winogrande, wiqa_*
Summarization / Dialogue	aeslc, dream_*, gem_wiki_lingua_english_en, samsum, gigaword
Data-to-Text / Structured Generation	gem_common_gen, gem_dart, gem_e2e_nlg, gem_web_nlg_en, wiki_bio_*
Translation	para_crawl_enes, wmt14_translate_fr-en, wmt16_translate_cs-en, wmt16_translate_de-en, wmt16_translate_fi-en, wmt16_translate_ro-en, wmt16_translate_ru-en, wmt16_translate_tr-en
Miscellaneous / Preprocessing	definite_pronoun_resolution, fix_punct, huggingface, math_dataset_algebra__linear_1d, opinion_abstracts_*, true_case, word_segment

Table 5: Full list of Flan-v2 datasets used for LoRA training, grouped by task category. For brevity, "*" denotes multiple variants included in the collection.



(a) Norm (b) Entropy

Figure 8: Performance of LoGo using (a) norm and (b) entropy across datasets with different tokens for signal extraction, first, average, and last. Here, average denotes the mean signal values across all tokens. The last token serves as the default setting of our method.



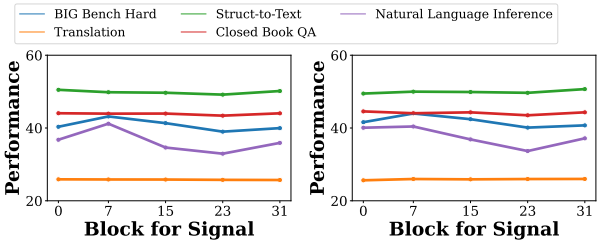
(a) Norm (b) Entropy

Figure 9: Performance of LoGo using (a) norm and (b) entropy across datasets with different numbers of selected modules.

slightly higher performance across most datasets, supporting its use as the default configuration in our method.

Number of Selected Modules We analyze the effect of varying the number of selected modules k in LoGo by evaluating performance with $k \in 3, 5, 10, 20$. Fig. 9 presents the results for both norm- and entropy-based scoring. Overall, performance improves as the number of selected modules increases, but the gains are relatively modest. This suggests that LoGo is not highly sensitive to the exact choice of k: even with only a few modules, it achieves performance close to the larger settings. Such robustness further highlights the practicality of LoGo, as it enables efficient operation with a small number of modules while retaining strong performance.

Block for Signal Extraction To assess the sensitivity of LoGo to the layer from which signals are extracted, we vary the target Transformer block used for signal computation. Specifically, we extract projection-based signals from the 0-th, 7-th, 15-th, 23-rd, and 31-st blocks. Results in Fig. 10

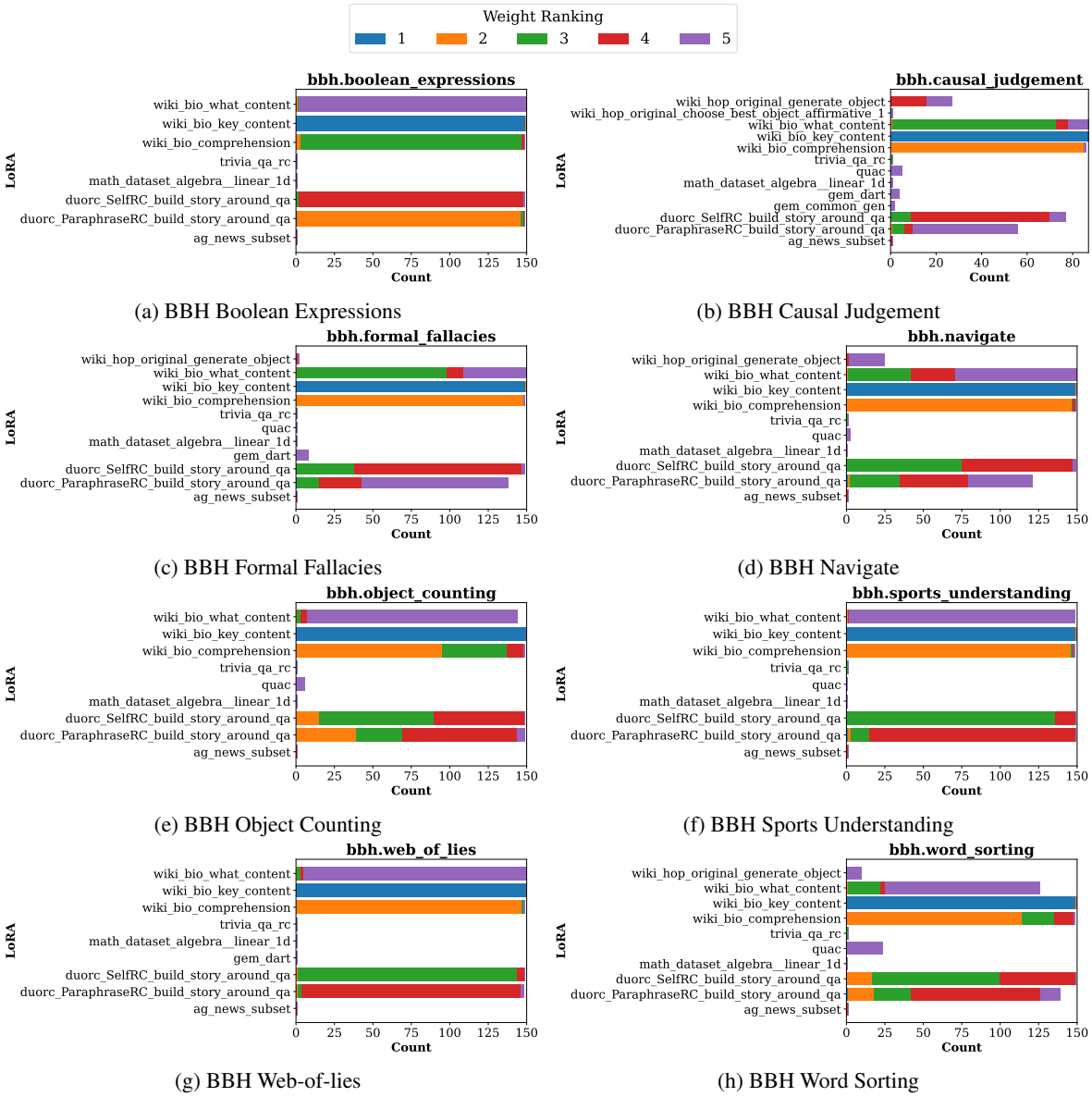


(a) Norm (b) Entropy

Figure 10: Performance of LoGo using (a) norm and (b) entropy across datasets with different target block for signal extraction.

show minor variations in performance across layers, indicating that LoGo is not sensitive to the specific block chosen for signal extraction. This suggests that task-relevant activation patterns are distributed across multiple layers, and that LoGo can robustly estimate adapter relevance from various depths without requiring careful layer tuning.

D.2 Selected LoRAs



(a) BBH Boolean Expressions

(b) BBH Causal Judgement

(c) BBH Formal Fallacies

(d) BBH Navigate

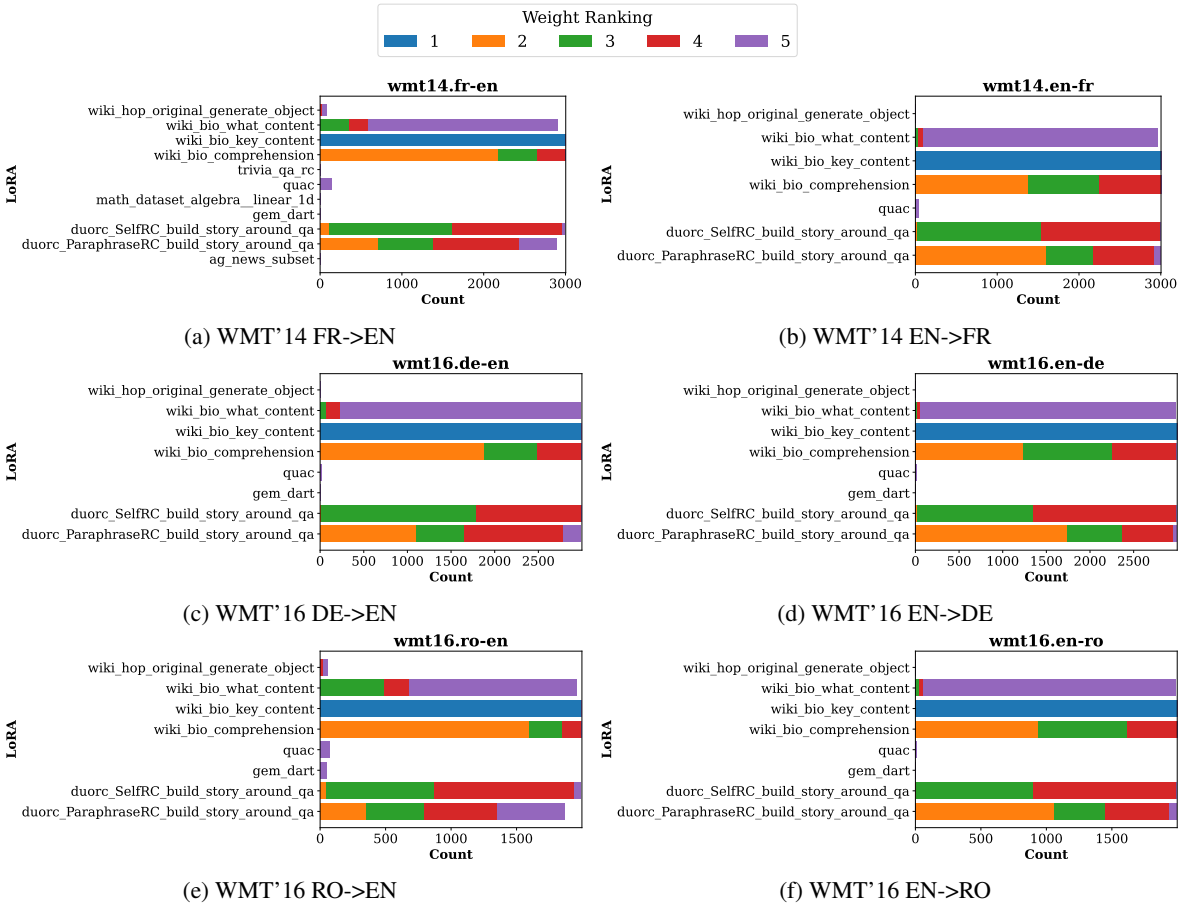
(e) BBH Object Counting

(f) BBH Sports Understanding

(g) BBH Web-of-lies

(h) BBH Word Sorting

Figure 11: The LoRA selection count by LoGo with Llama-3.1-8B model for BIG Bench Hard datasets.

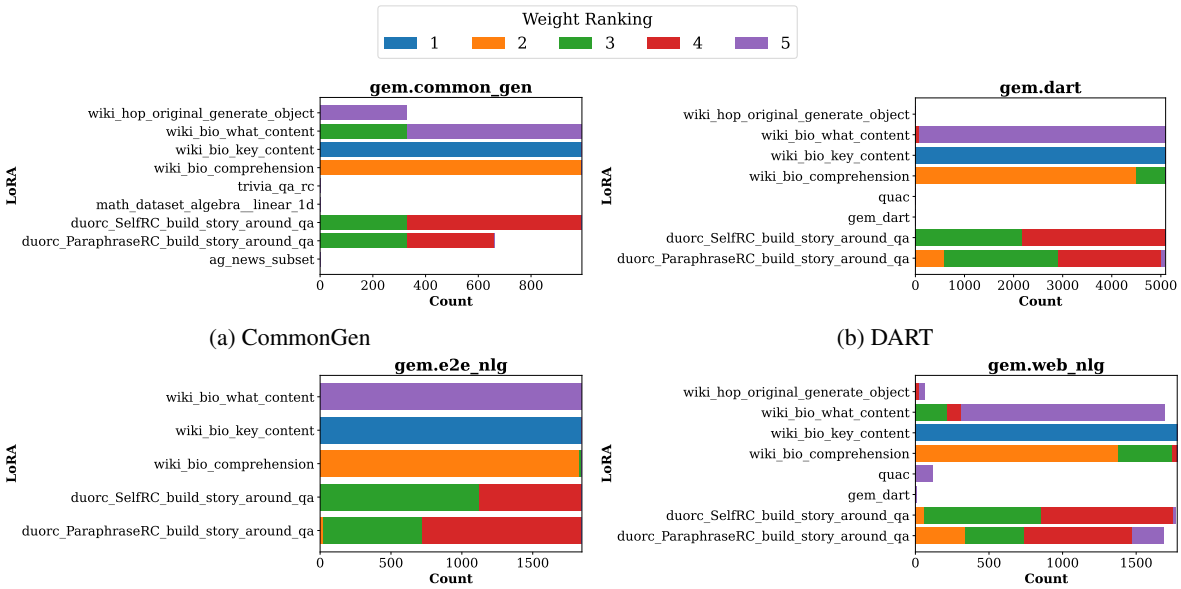


(a) WMT'14 FR->EN (b) WMT'14 EN->FR

(c) WMT'16 DE->EN (d) WMT'16 EN->DE

(e) WMT'16 RO->EN (f) WMT'16 EN->RO

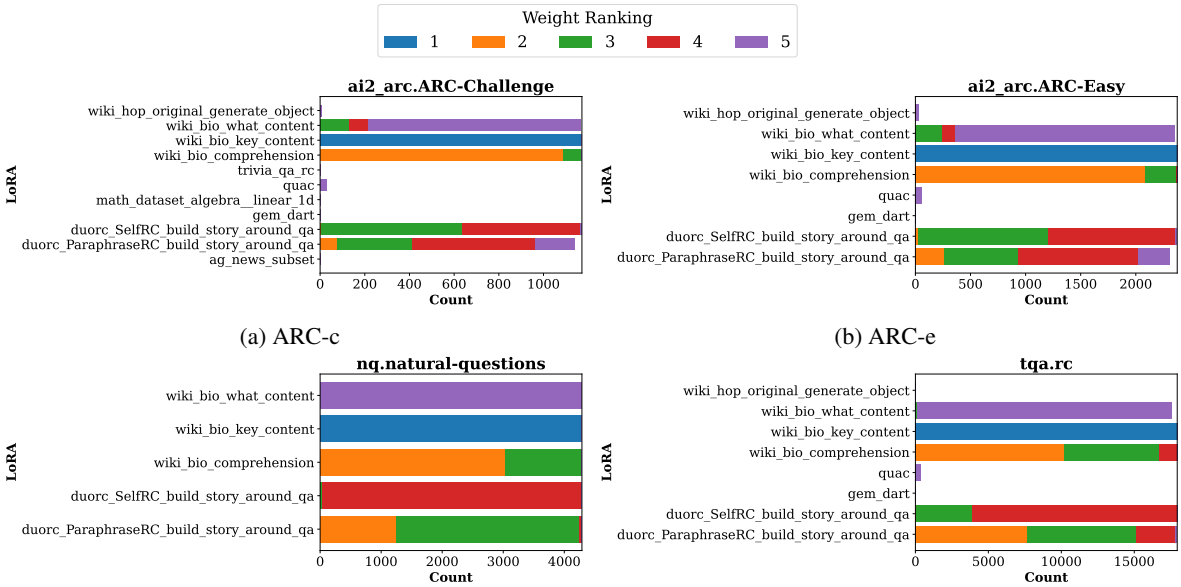
Figure 12: The LoRA selection count by LoGo with Llama-3.1-8B model for translation datasets.



(a) CommonGen (b) DART

(c) E2ENLG (d) WebNLG

Figure 13: The LoRA selection count by LoGo with Llama-3.1-8B model for struct-to-text datasets.



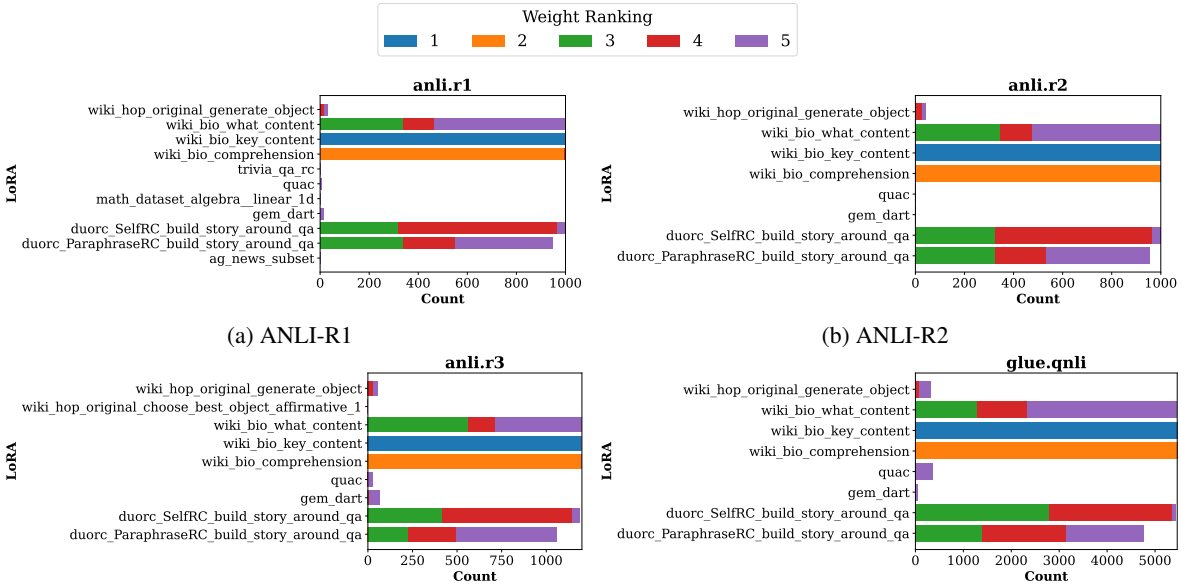
(a) ARC-c

(b) ARC-e

(c) Natural Questions

(d) Trivia QA

Figure 14: The LoRA selection counts by LoGo with Llama-3.1-8B model for closed-book QA datasets.



(a) ANLI-R1

(b) ANLI-R2

(c) ANLI-R3

(d) QNLI

Figure 15: The LoRA selection counts by LoGo with Llama-3.1-8B model for NLI datasets.