# ShapLoRA: Allocation of Low-rank Adaption on Large Language Models via Shapley Value Inspired Importance Estimation

Yi Zhao[1]*, Qinghua Yao[2], Xinyuan song[3], Wei Zhu[4]
[1]Singapore Management University, [2]University of Pennsylvania, [3]Emory University, [4]University of Hong Kong
yizhao@smu.edu.sg,
yqinghua@seas.upenn.edu, xinyuan.song@emory.edu, wzhu91@connect.hku.hk

Low-rank adaption (LoRA) is a representative method in the field of parameter-efficient fine-tuning (PEFT), and is key to Democratizating the modern large language models (LLMs). The vanilla LoRA is implemented with uniform ranks, and the recent literature have found that properly allocating ranks on the LLM backbones results in performance boosts. However, the previous rank allocation methods have limitations since they rely on inexplanable and unreliable importance measures for the LoRA ranks. To address the above issues, we propose the ShapLoRA framework. Inspired by the explanable attribution measure Shapley Value, we combine the sensitivity-based measures with the idea of coalitions in the collaborative games among LoRA ranks, and propose a more explainable importance measure called Shapley sensitivity. In addition, we optimize the workflow of the existing works by: (a) calculating Shapley sensitivity on a separate validation set; (b) Setting up the allocating-retraining procedures for fair comparisons. We have conducted experiments on various challenging tasks, and the experimental results demonstrate that our ShapLoRA method can outperform the recent baselines with comparable tunable parameters.[2]

## 1. Introduction

Large language models (LLMs) have demonstrated remarkable capabilities, achieving state-of-the-art (SOTA) performance across diverse natural language processing (NLP) tasks [1, 2] as well as specialized evaluation benchmarks [3, 4], including domain-specific question answering, mathematical reasoning, safety alignment, and instruction comprehension. While LLMs increasingly serve as general-purpose problem solvers, fine-tuning remains critical for optimizing inference efficiency and tailoring the style or tone of model outputs. Recent developments, such as OpenAI's fine-tuning APIs for GPT-3.5-turbo and GPT-4[3], underscore its practical relevance. Nevertheless, full-parameter fine-tuning of LLMs is often prohibitively expensive, demanding substantial GPU memory and computational resources not only during training but also during deployment. To address this, parameter-efficient fine-tuning (PEFT) methods [5, 6] have gained prominence, enabling effective adaptation with tunable parameters typically accounting for less than 1% of the total model weights while drastically reducing training costs.

Parameter-Efficient Fine-Tuning (PEFT) methods have become indispensable for adapting large language models (LLMs). Among these, Low-Rank Adaptation (LoRA) [7] has emerged as a particularly effective reparameterization-based approach, achieving widespread adoption in LLM fine-tuning [8–10]. However, the vanilla LoRA applies a uniform LoRA rank allocation setting across the Transformer layers and linear modules, which leaves room for improvements. Recent, a series

*Corresponding author
[2]Codes and fine-tuned models will be open-sourced to facilitate future research.
[3]https://openai.com/blog/gpt-3-5-turbo-fine-tuning-and-api-updates

Second Conference on Parsimony and Learning (CPAL 2025).

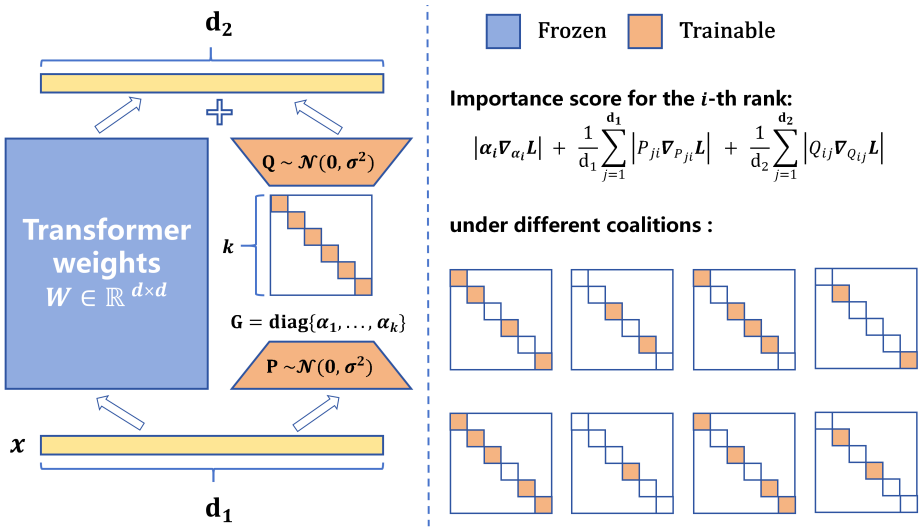



Figure 1: Schematic illustration of our ShapLoRA framework. (**Left**): ShapLoRA follows LoRA and AdaLoRA to update the weight matrix $W$ by fine-tuning the low-rank incremental matrix $\Delta W = PGQ$ with a SVD formation. And LoRA rank $i$ is represented by a tuple $(G_{i,i}, P_{:,i}, Q_{i,:})$ of parameters. (**Right**): ShapLoRA prunes the less important LoRA rank $k$ by fixing $G_{k,k}$ to zero. Here the importance score IPT(k) for each LoRA rank $k$ is calculated as the average sensitivity score of the parameters in rank $k$. Different from previous works, IPT(k) is calculated under a random subset of coalitions in which each LoRA rank is considered as a player in a collaborative game. A coalition is formulated by randomly masking some of the elements of $G$ to zeros.

of LoRA variants [11–15] propose different approaches to allocate or prune the uniform distributed LoRA ranks adaptively on the given task and LLM backbone, achieving performance boosts on the downstream tasks. However, the existing methods rely on un-reliable and in-explainable LoRA rank importance measurements, thus they still can not fully exploit the potential of LoRA fine-tuning.

To address the above issues, we propose the novel Shapley Value inspired Low-Rank Adaptation framework (ShapLoRA). To address the shortcomings of the sensitivity-based importance measure, we propose to put this measure under different coalitions in the collaborative games among LoRA ranks, inspired by game theory based attribution method Shapley Value [16]. This results in a novel and more explainable importance measure, which is named as Shapley sensitivity, to pay homage to Shapley Value. In addition, we optimize the workflow of the existing works by: (a) conducting LoRA rank pruning once based on the Shapley sensitivity scores on a separate validation set instead of the training set. (b) Setting up the allocating-retraining procedures for fair comparisons among different PEFT methods.

We conduct comprehensive experiments across diverse challenging tasks, including question-answering tasks, math problem solving tasks, and a collection of natural language processing tasks. Our approach consistently surpasses strong parameter-efficient fine-tuning (PEFT) baselines, including state-of-the-art LoRA variants, under comparable trainable parameter budgets. The experimental results also demonstrate the general applicability of our ShapLoRA framework through training and inference efficiency analysis. Our contributions are summarized as follows:

- we propose ShapLoRA, a novel framework designed to adaptively adjust the LoRA rank allocation on the LLM backbone, resulting in improved fine-tuning quality.
- The core of our ShapLoRA framework is the our proposed Shapley sensitivity, an LoRA rank importance measure which combines the gradient-based sensitivity measure and game theory.
- Through comprehensive empirical evaluations and in-depth analysis, we demonstrate that our proposed ShapLoRA framework delivers high-performance LoRA rank allocation settings that consistently surpass baseline methods under equivalent tunable parameter budgets.

## 2. Related works

Due to limited length, we put more literature reviews on parameter-efficient fine-tuning to Appendix A.

### 2.1. The LoRA method and its variants

The third line of works, which our work belongs to, address the issue of allocating LoRA ranks among the linear modules across different layers of the Transformer-based LLM backbones. AdaLoRA [11] expresses the low-rank multiplication of LoRA in the form of singular value decomposition (SVD), and it identifies the most important ranks by a sensitivity-based importance score. SoRA [12] prunes abundant LoRA ranks by imposing a $l_0$ norm and optimizing with proximal gradient descent. SaLoRA [13] prunes the LoRA ranks via the Lagrange multiplier method. ALoRA [15] address the importance estimation of LoRA ranks via an noval heuristic measure. AutoLoRA [14] proposes to associate each LoRA rank with a selection variable, and updates these variables during the fine-tuning procedure. Then the pruning of LoRA ranks relies on the magnitude of these learned selection variables. AutoLoRA applies the idea of differentiable architecture search [14] into LoRA rank allocation, and these selection variables are referred to as the architectural parameters by [17]. Despite these recent efforts, we believe issues still need to be investigated for LoRA rank allocation: the existing works replies on flawed importance estimations which can not reliably reflect the contribution of each LoRA rank. Our work complements the existing literature by proposing a novel LoRA rank importance measurement.

## 3. Background

**Transformer model** Currently, the most widely used open-sourced large language models adopt the stacked Transformer architecture [18]. The transformer block is primarily constructed using two key submodules: a multi-head self-attention (MHA) layer and a fully connected feed-forward (FFN) layer. Denote the input sequence's length as $l$, the hidden states' dimension as $d_{model}$, and the dimension at the FFN module as $d_{ffn}$. The MHA is given as follows:[4]

$$\text{softmax}\left(\frac{QK}{\sqrt{d_{model}}}\right)V, \tag{1}$$

where $Q = xW^Q$, $K = xW^K$, $V = xW^V$, $x \in \mathbf{R}^{l\times d_{model}}$ is the input tensor. $W^Q, W^K, W^V \in \mathbf{R}^{d_{model}\times d_{model}}$ are the query, key, and value projection layers (denoted as the Query, Key, and Value modules, or the Q, K, V modules). The FFN module consists of linear transformations and an activation function $g^{ffn}$ such as ReLU or GELU [19]. Take the FFN module in the LlaMA-2 models [20] as example:

$$(g^{ffn}(G) * U)W^D, \tag{2}$$

where $G = xW^G$, $U = xW^U$, $W^G, W^U \in \mathbf{R}^{d_{model}\times d_{ffn}}$ (denoted as Gate and Up module, or the G and U modules). The number of linear modules in a Transformer block is denoted as $N_{mod} > 0$. Thus, in LlaMA-2, $N_{mod} = 7$.

**Low-rank adaptation (LoRA)** For any Transformer's linear module $m \in$ {Q, K, V, O, G, U, D} on the $l$-th ($1 \le l \le L$, $L > 0$ denoting the number of Transformer layers in the LLM), AdaLoRA [11] formulates LoRA in a singular value decomposition (SVD) format:

$$x^{'} = xW^{l,m} + b^{l,m} + xP^{l,m}\Lambda^{l,m}Q^{l,m}, \tag{3}$$

where $P^{l,m} \in \mathbb{R}^{d_1\times r}$ and $Q^{l,m} \in \mathbb{R}^{r\times d_2}$ are the left/right singular vectors, $\Lambda^{l,m} \in \mathbb{R}^{r\times r}$ is the diagonal matrix contains the singular values $\{\lambda_i^{l,m}\}_{i=1}^r$. These three matrices contain the tunable parameters for the LoRA module. $\Lambda^{l,m}$ is initialized with zero while $P^{l,m}$ and $Q^{l,m}$ adopt a random Gaussian initialization to ensure $P^{l,m}\Lambda^{l,m}Q^{l,m}$ is a zero matrix at the beginning of training. Under

[4] We omit the multi-head setting for simplicity of illustrations.

Equation 3, the $i$-th rank of LoRA $(l, m)$ contains the $i$-th singular value and vectors, and is denoted as $\mathcal{G}_i^{l,m} = (\lambda_i^{l,m}, P_{*,i}^{l,m}, Q_{i,*}^{l,m})$. Note that during the training process of AdaLoRA, LoRA rank pruning is conducted by setting some of the singular values to zero step-by-step.

# 4. Method

## 4.1. Motivation

We now reflect on the previous representative works on LoRA rank allocation. AdaLoRA [11] first consider re-arrage the rank distributions of LoRA modules on the LLM backbone. It achieve this objective by first initialize all the LoRA modules with a large number of ranks, and prune the less important ranks gradually along with the training procedure. AdaLoRA utilize a sensitivity based importance score [21],

$$\text{ipt}(w) = \|w \nabla_w \mathcal{L}\| \tag{4}$$

which measures how much the training loss will change if the LoRA parameters change. However, [22] pointed out that this importance measure is unreliable, since it only considers how one parameter change affects the model under the hypothesis that no other parameter changes occur, and have not consider its importance under different model statuses.

AutoLoRA [14] builds upon the methodology of differentable neural architecture search and bi-level optimization [17]. It considers each LoRA rank as a neural network operation and assigns a learnable architectural parameter. Its objective is to select the best LoRA architecture, which relies on the learned architectural parameters' values as the importance scores. SoRA [12] and SaLoRA [13] are similar to AutoLoRA except that the architectural parameters are learned with a normal optimization procedure on the training set [23]. The LoRA ranks with higher architectural weights are kept while others are pruned. However, as pointed out by [24], the architectural parameters can not reliably reflect the quality or importance of the LoRA ranks.

To summarize, to further enhance the performance of LoRA under rank allocation, we need to find an importance measure that is reliable and explainable. Thus, we draw inspiration from Shapley Value [16]. Shapley Value considers the collaborations among a group of players. In the context of this work, a player in the game is a LoRA rank. For player $k$ in the game, denotes all the other permutations of the other players as $\mathcal{S}_k$, then player $k$'s Shapley Value $\Phi_k$ is defined as:

$$\Phi_m = \frac{1}{|\mathcal{S}_k|} \sum_{A \in \mathcal{S}_k} V(A \cup \{k\}) - V(A), \tag{5}$$

where $V()$ denotes the performance metric for any coalition $A \in \mathcal{S}_k$. Intuitively, the calculation of the player $k$'s Shapley Value involves evaluating its contributions across different coalitions, that is, evaluating this player under different scenarios. However, Shapley Value is extremely time consuming, prohibiting its applications in deep learning. However, the core idea of Shapley Value could be the guide for our work.

## 4.2. Shapley Sensitivity

Now we introduce the core our ShapLoRA method, Shapley Sensitivity. This method combines the core idea of Shapley Value with the gradient-based sensitivity method [21]. In order to evaluate the LoRA rank $\mathcal{G}_i^{l,m}$ from LoRA $(l, m)$ in the style of Shapley Value, we put it under different coalitions by randomly masking the singular values of LoRA modules. For the LoRA rank $\mathcal{G}_i^{l,m}$, all the other LLM LoRA ranks' permutations are denoted as $\mathcal{S}_i^{l,m}$. If one wants to evaluate the LLM's performance under a permutation $S \in \mathcal{S}_i^{l,m}$, we need to exclude all the LoRA ranks not in $S$ to zeros:

$$\lambda_{i'}^{l',m'} = \begin{cases} \lambda_{i'}^{l',m'} & \text{if } \mathcal{G}_{i'}^{l',m'} \in S \\ 0 & \text{if } \mathcal{G}_{i'}^{l',m'} \notin S \end{cases}. \tag{6}$$

Then, we calculate the importance score $\text{SAN}(\mathcal{G}_i^{l,m}|S)$ of LoRA rank $\mathcal{G}_i^{l,m}$ under permutation $S$:

$$\text{SAN}(\mathcal{G}_i^{l,m}|S) = \text{ipt}(\lambda_i^{l,m}) + \frac{1}{d_1}\sum_{j=1}^{d_1}\text{ipt}(P_{j,i}^{l,m}) + \frac{1}{d_2}\sum_{j=1}^{d_2}\text{ipt}(Q_{i,j}^{l,m}), \tag{7}$$

where $\text{ipt}(w)$ denotes the gradient-based sensitivity score for a model parameter:

$$\text{ipt}(w) = \|w\nabla_w\mathcal{L}\|, \tag{8}$$

in which $\mathcal{L}$ denotes the loss objective during fine-tuning. Then, the Shapley sensitivity $\text{SAN}(\mathcal{G}_i^{l,m})$ of LoRA rank $\mathcal{G}_i^{l,m}$ is given by:

$$\text{SAN}(\mathcal{G}_i^{l,m}) = \frac{1}{|\mathcal{S}_i^{l,m}|}\sum_{S\in\mathcal{S}_i^{l,m}}\text{SAN}(\mathcal{G}_i^{l,m}|S). \tag{9}$$

Note that although more efficient than Shapley Value, the above equation for calculating Shapley sensitivity is still prohibitively computation-consuming. The next subsection will introduce an approximation to $\text{SAN}(\mathcal{G}_i^{l,m})$.

### 4.3. Workflow of ShapLoRA

We now describe the complete workflow of our ShapLoRA method during the rank allocation stage, which is based on our proposed Shapley sensitivity score. The training set for a downstream task is denoted as $\mathcal{D}_{train}$, and the validation set is denoted as $\mathcal{D}_v$.

Initially, all the LoRA modules with equal ranks $r_{init}$ are installed on the LLM backbone. Thus the initial total number of ranks is $R^{init} = N_{mod} * L * r_{init}$. And our targeted total number of LoRA ranks is $R^{target} > 0$. These LoRA parameters are fine-tuned for $K_1 > 0$ epochs to ensure convergence. Then the LoRA ranks' Shapley sensitivity scores will be calculated on $\mathcal{D}_v$, and $R^{prune} = R^{init} - R^{target}$ ranks with lower scores will be pruned, and we obtain the rank allocation configurations.

Since Equation 9 is impractical, we need to provide an approximation to strike a balance between efficiency and performance. Denote $\mathcal{S}_{all}$ as all the permutations of the LoRA ranks. Then a subset $\mathcal{S}_{sub}$ of size $N_3 > 0$ is drawn from $\mathcal{S}_{all}$. [5] And the the Shapley sensitivity scores for any LoRA rank $\mathcal{G}_i^{l,m}$ is given by:

$$\text{SAN}(\mathcal{G}_i^{l,m}) \approx \frac{\sum_{S\in\mathcal{S}_{sub}}\text{SAN}(\mathcal{G}_i^{l,m}|S)}{|\mathcal{S}_{sub}|}. \tag{10}$$

## 5. Experiments

In this section, we conduct extensive experiments to evaluate our ShapLoRA method.

### 5.1. Datasets and evaluation metrics

We compare our approach to the baselines on a collection of challenging tasks: (a) five benchmark common-sense question-answering tasks, ARC-e and ARC-c [25], OBQA [26], PIQA [27], BoolQ [28]. (b) two math reasoning tasks, AQuA [29] and GSM8k [30]. (c) MT-Bench [31], MMLU [32], and BBH [33]. Since these tasks provide no training data, we utilize the UltraChat [34] dataset for instruction tuning. (d) A collection of natural language processing (NLP) or natural language generation (NLG) tasks, including SST-2, RTE, QNLI from the GLUE benchmark [35], a conditional generation task E2E [36], and a SQL generation task WikiSQL [37].

Dataset introductions, statistics and evaluation metrics are introduced in Appendix B.

[5] Once a $S \in \mathcal{S}_{all}$ is drawn, we also draw its complementary $S^{'}$ to $\mathcal{S}_{sub}$. That is $S \cup S^{'}$ contains all the LoRA ranks. This ensures that all the LoRA ranks will be masked with equal times.

| Method | Tunable Params | ARC-e (acc) | ARC-c (acc) | BoolQ (acc) | OBQA (acc) | PIQA (acc) | AQuA (acc) | GSM8k (acc) | Avg. |
|---|---|---|---|---|---|---|---|---|---|
| *Baselines* | | | | | | | | | |
| Parallel-Adapter | 20.9M | 72.4 | 54.2 | 75.3 | 76.3 | 69.8 | 45.6 | 56.4 | 64.3 |
| Learned-Adapter | 21.1M | 73.1 | 54.4 | 74.9 | 78.4 | 75.6 | 48.3 | 58.9 | 66.2 |
| P-tuning v2 | 21.0M | 68.5 | 51.3 | 71.2 | 76.1 | 66.2 | 39.63 | 51.1 | 60.6 |
| IAPT | 20.9M | 75.1 | 54.7 | 77.8 | 79.2 | 77.3 | 43.6 | 55.8 | 66.2 |
| BitFit | 25.2M | 72.3 | 54.1 | 76.4 | 77.2 | 76.6 | 41.8 | 51.7 | 64.3 |
| $(IA)^3$ | 23.1M | 73.1 | 54.6 | 77.2 | 78.1 | 75.4 | 43.2 | 53.4 | 65.0 |
| SSP | 80.6M | 75.2 | 57.6 | 79.6 | 79.5 | 79.7 | 45.9 | 61.8 | 68.5 |
| LoRA | 22.5M | 74.4 | 57.2 | 78.8 | 81.1 | 81.4 | 46.6 | 61.1 | 68.7 |
| AdaLoRA | 23.2M | 75.1 | 57.9 | 79.2 | 81.4 | 82.1 | 47.6 | 61.7 | 69.3 |
| AutoLoRA | 23.1M | 76.9 | 59.6 | 80.3 | <u>81.7</u> | 82.5 | 47.9 | 61.3 | 70.0 |
| MOELoRA | 29.9M | 77.5 | <u>60.2</u> | 81.4 | 81.7 | 82.4 | <u>48.3</u> | 62.3 | 70.5 |
| DoRA | 22.6M | <u>77.9</u> | 59.8 | <u>81.7</u> | 81.6 | <u>82.7</u> | 47.9 | <u>62.6</u> | <u>70.6</u> |
| *Our proposed methods* | | | | | | | | | |
| **ShapLoRA** | **22.8M** | **79.3** | **61.1** | **82.8** | **82.9** | **84.5** | **49.7** | **64.4** | **72.1** |

Table 1: The Overall comparison of different PEFT methods. The backbone model is LLaMA-3 8B. Bold and underline indicate the best and second-best results.

## 5.2. Baselines

We compare our ShapLoRA framework with the current SOTA PEFT baseline methods.

**LoRA and its variants** we consider the following LoRA variants as baselines: (a) the original LoRA [7]; (b) AdaLoRA [11], which adaptively adjust the LoRA parameters among different Transformer modules. (c) AutoLoRA [14], which utilize the bi-level optimization method [17] to learn the LoRA ranks' importance scores. (d) MOELoRA [38], which considers each LoRA module as a mixture of single-rank LoRA experts. (e) DoRA [39].

**Other PEFT methods** We also consider the most recent PEFT methods: (a) Parallel-Adapter proposed by He et al. [40]; (b) Learned-Adapter [5]. (c) P-tuning v2 [41]. (d) IAPT [42]. (e) BitFit [43]. (f) $(IA)^3$ [44], which multiplies learnable vectors to the hidden states in different modules of the Transformer layer. (g) SSP [45].

The baselines are implemented using their open-sourced codes. We only adjust the hyper-parameters related to tunable parameter numbers to fairly compare the baseline methods and our ShapLoRA method. The hyper-parameter settings for the baselines are detailed in Appendix D.

## 5.3. Experiment Settings

**Computing infrastures** We run all our experiments on NVIDIA A40 (48GB) GPUs.

**Pretrained backbones** The main experiments use the most recent open-sourced LLM, LlaMA-3 8B released by Meta [46] as the pretrained backbone model. In the ablation studies, we will also use the distilled Qwen 3B models and distilled LlaMA 8B models from Deepseek R1 [47].

**Prediction heads and decoding** When fine-tuning a LLM, we only consider the supervised fine-tuning (SFT) setting [48]. After receiving a prompt or instruction, all the predictions are generated using the language modeling head (LM head). No additional prediction heads are installed for making categorical or numerical predictions. For decoding during inference, we use beam search with beam size 3.

**Hyper-parameter settings for ShapLoRA** Our ShapLoRA divide the whole workflow into two stages, the LoRA rank allocation stage and the final fine-tuning stage. During the LoRA rank allocation stage, we add LoRA modules with rank $r_init = 16$ at each linear module of the Transformer block, and use the given task's training set to fine-tune the LoRA parameters till convergence. Then, each LoRA rank is evaluated by our Shapley sensitivity measure (Equation 10). In Equation 10, we set the size of $\mathcal{S}_{sub}$ to 90 by randomly masking each LoRA rank with a binomial distribution with

parameter in { 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 } repeatedly for 5 times. Then, the LoRA ranks that received lower Shapley sensitivity scores are pruned to meet the targeted average LoRA rank budget $R^{target} = \frac{R^{init}}{2}$.

During the final fine-tuning stage, all the LoRA modules are randomly initialized according to the allocation setting delivered by the previous stage.

The settings for training on both stages are specified in Appendix D.

**Reproducibility** We run each task under five different random seeds and report the median performance on the test set of each task.

Due to limited length, other experimental settings for the baseline methods and the training procedure are put in Appendix D.

## 5.4. Main results

**Results for QA and math tasks.** In this setup, We compare ShapLoRA with baseline PEFT methods by employing these methods in fine-tuning on a challenging downstream task. The experimental results on the five commonsense reasoning QA tasks and two math reasoning tasks are presented in Table 1. We present the number of tunable parameters in the second column. Table 1 reveals that our ShapLoRA method outperforms the baseline methods across all seven tasks, with comparable tunable parameters. In particular, ShapLoRA outperforms the previous SOTA LoRA-based baselines like AdaLoRA, AutoLoRA, DoRA, and MOELoRA with comparable or less tunable parameters. These results demonstrate that our method excels at downstream task adaptation of large language models.

**Results for general-purpose instruction tuning.** After fine-tuning the large language model (LLM) (pretrained version) on the UltraChat dataset [34] using our proposed ShapLoRA method as well as AutoLoRA and MOELoRA[6], we evaluate performance across three challenging benchmarks: MT-Bench [31], MMLU [32], and BBH [33]. For MT-Bench, we report the average GPT-4 evaluation score (denoted as gpt4-score), while comprehensive results are detailed in Table 2. Consistent with earlier findings (Table 1), ShapLoRA achieves superior performance compared to MOELoRA across all benchmarks. These results underscore ShapLoRA's efficacy in improving instruction-tuning quality for pretrained versions of the LLMs, highlighting its potential as a robust alternative to existing parameter-efficient adaptation strategies.

**Results on the GLUE and NLG tasks** The experimental results on the three classification tasks and two NLG tasks are presented in Table 6. Table 6 agains proves that our ShapLoRA method outperforms the strong baseline methods across all five tasks.

## 5.5. Ablation studies and further analysis

**Analysis of the training and inference efficiency** We now use the BoolQ task to analyze how much additional memory and time our ShapLoRA requires compared to the MOELoRA and AutoLoRA methods. Table 3 presents the peak memory cost (in GiB) and time cost till obtaining the final LoRA parameters (in hours). All the methods requires comparable memory costs since the majority of the memory cost is due to the LLM backbone. MOELoRA requires few training hours since it does not require a two-stage workflow. AutoLoRA requires to learn to selection variables at the first stage with the help of bi-level optimization, which is time-costing. And it need to retrain the LoRA parameters after pruning. Our method's workflow is similar to AutoLoRA, and it turns out that the calculation of our Shapley sensitivity scores does not lead to excessive time costs. Considering its superior performance compared to the baselines, our method is practical since its training costs is within a reasonable range.

[6]Due to limited resources, we now only provide the results for our ShapLoRA method and two representative and strong baselines on these LLM evaluation benchmarks. We will provide results for more baselines in the updated version.

| Method | MT-Bench gpt4-score | MMLU acc | BBH acc |
|---|---|---|---|
| MOELoRA | 7.31 | 56.6 | 47.8 |
| AutoLoRA | 7.39 | 57.1 | 47.6 |
| **ShapLoRA** | **7.56** | **58.7** | **48.7** |

Table 2: General-purpose instruction tuning performance.

| Method | Time cost (hours) | Memory cost (GiB) |
|---|---|---|
| MOELoRA | 2.1 | 18.2 |
| AutoLoRA | 5.3 | 18.6 |
| **ShapLoRA** | 4.8 | 17.9 |

Table 3: Time and memory cost on BoolQ.

The inference efficiency analysis is presented in Table 8 in Appendix F, which shows that the LoRA configurations obtained by ShapLoRA has the comparable decoding speed with baselines.

**Visualization and analysis of the ShapLoRA's importance score distributions and rank allocation settings** We present the LoRA ranks' Shapley sensitivity scores (normalized per linear module) at the 8th, 16th, 24th, and 32nd layer of LlaMA-3 8B as a heatmap in Figure 3 and 4 corresponding to the BoolQ and PIQA task, respectively in Appendix G. And we present the obtained LoRA rank configurations after pruning in Figure 5. We can observe that:

- Different downstream tasks delivers different rank allocation configurations in our ShapLoRA framework, demonstrating the task specificity of LoRA fine-tuning.
- However, similarity in LoRA rank allocations across different tasks can be observed. We can observe from Figure 5 that more LoRA ranks are pruned from the Query and Key modules, while the Value module keeps most of the LoRA ranks.
- The LoRA importance distributions across different modules and Transformer layers are different. Intuitively, different modules at Transformer layers play different roles, thus requiring different quantities of LoRA parameters.

**Ablation on the ShapLoRA framework** In order to further demonstrate the effectiveness of ShapLoRA, and the appropriateness of its workflow designs, we now conduct a series of ablation studies on our workflow. We now consider the following variants of the ShapLoRA framework: (a) ShapLoRA-1 substitutes our proposed Shapley sensitivity score to the original sensitivity score [21]. That is, ShapLoRA-1 is equivalent to AdaLoRA with just one pruning step. (b) ShapLoRA-2 substitutes our proposed Shapley sensitivity score to the magnitude-based score [49]. (c) ShapLoRA-3 considers a pruning schedule similar to AdaLoRA, that is, pruning 4 LoRA ranks once every 25 steps. (d) ShapLoRA-4 considers calculating the Shapley sensitivity score on the training set. (e) ShapLoRA-5 reduces the times of random masking to 18 times. (f) ShapLoRA-6 increases the times of random masking to 900 times.

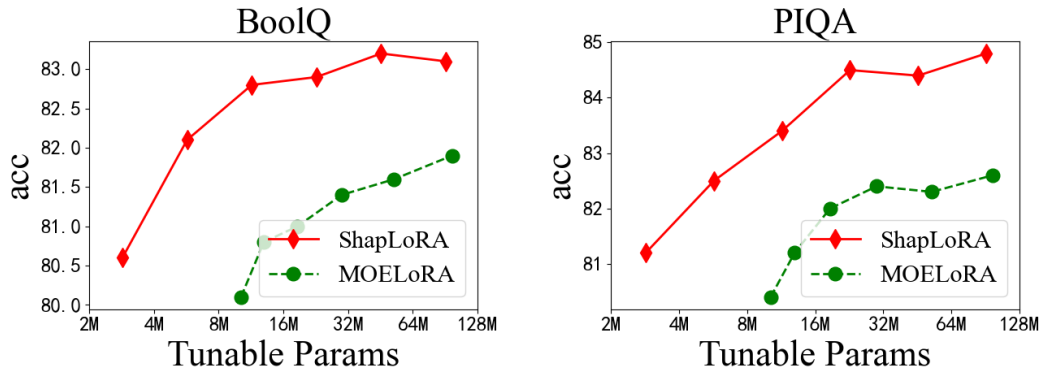
The ablation experiments on the BoolQ, PIQA, and MMLU are presented in Table 4. The results show that ShapLoRA under the default settings (as in Table 1) outperforms or performs comparable to the six variants. In addition: (a) Comparing ShapLoRA to ShapLoRA-1 and ShapLoRA-2 demonstrates the effectiveness of our Shapley sensitivity score in identifying important LoRA ranks. (b) ShapLoRA-3 performs comparable to ShapLoRA, showing that a fine-grained pruning schedule is not necessary under our framework. (c) ShapLoRA-4 performs slightly worse than ShapLoRA, showing that it is better to determine which LoRA ranks to prune on a separate validation set to avoid receiving biased estimations of importance on the overfitted training data. We believe that this is one of the major reason why AutoLoRA outpeforms AdaLoRA. (d) Comparing ShapLoRA with ShapLoRA-5 and ShapLoRA-6 demonstrates that the random masking setting for calculating the Shapley sensitivity strikes a good balance between performance and efficiency.

**Comparisons under different budgets of tunable parameters** In our main experiments (Table 1), we set the targeted average LoRA ranks in the ShapLoRA setting to $r_{target} = 8$. Now we change the budget of tunable parameters for ShapLoRA by modifying the $r_{target}$ to {1, 2, 4, 16, 32}. We also alter the MOELoRA method's tunable parameter numbers accordingly. The experimental results on the BoolQ and PIQA tasks are presented in Figures 2a and 2b. The results show that under different

| Method | BoolQ (acc) | PIQA (acc) | MMLU (acc) |
|---|---|---|---|
| ShapLoRA | 82.8 | 84.5 | 58.7 |
| ShapLoRA-1 | 81.8 | 83.6 | 57.9 |
| ShapLoRA-2 | 81.0 | 82.1 | 56.8 |
| ShapLoRA-3 | 82.8 | 84.6 | 58.6 |
| ShapLoRA-4 | 82.1 | 83.7 | 57.5 |
| ShapLoRA-5 | 82.2 | 83.3 | 57.6 |
| ShapLoRA-6 | 82.8 | 84.5 | 58.8 |

Table 4: The comparison of ShapLoRA's variants on the BoolQ, PIQA, and MMLU tasks.

tunable parameter budgets, our ShapLoRA method (a) can consistently outperform the MOELoRA method, and (b) is more robust to decreases in tunable parameter numbers.




(a) BoolQ (b) PIQA

Figure 2: Performances under different tunable parameter budgets. The $x$-axis represents the number of tunable parameters, and the $y$-axis represents the performance score.

**On the stability of the Shapley sensitivity scoring method** On a given LLM backbone, we examine the stability of our Shapley sensitivity score, which depends on random masking. We run the scoring procedure under three different random seeds and compare the resulting LoRA importance scores. The similarity between each pair of runs is measured using Spearman rank correlation. The three runs are not used in any earlier experiments.

The pairwise correlations are all close to one: the correlations involving Seed1 are 1.00 with itself, 0.99 with Seed2, and 0.96 with Seed3; the correlation between Seed2 and Seed 3 is 0.99. These high correlations show that the importance scores of the LoRA ranks remain stable across different random seeds, demonstrating that the Shapley sensitivity scoring used in ShapLoRA is robust to randomness in the masking process.

**Ablation on the pretrained backbones** Our main experiments are conducted on the LlaMA-3 8B model. To demonstrate the wide applicability of our method, we now conduct experiments on the distilled Llama-3 8B and Qwen2.5 3B from Deepseek R1 [47]. The results are reported in Table 7 in Appendix E. We can see that on these two backbones, our method can also outperform the baseline methods.

## 6. Conclusion

In this work, we introduced the ShapLoRA framework to enhance the efficiency of LoRA fine-tuning. To address the drawbacks in the recents works on LoRA rank allocation, we propose Shapley sensitivity, a novel LoRA rank importance measurement which combines the gradient-based sensitivity

and the idea of game-theory-based attribution methods. We have conducted experiments on various challenging tasks, and the experimental results demonstrate that our ShapLoRA method can outperform the recent baselines with comparable tunable parameters.

# References


[1] Chengwei Qin, Aston Zhang, Zhuosheng Zhang, Jiaao Chen, Michihiro Yasunaga, and Diyi Yang. Is chatgpt a general-purpose natural language processing task solver?, 2023.

[2] Wei Zhu, Xiaoling Wang, Huanran Zheng, Mosha Chen, and Buzhou Tang. PromptCBLUE: A Chinese Prompt Tuning Benchmark for the Medical Domain, October 2023.

[3] Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Jiayi Lei, et al. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models, 2023.

[4] Haonan Li, Yixuan Zhang, Fajri Koto, Yifei Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. Cmmlu: Measuring massive multitask language understanding in chinese, 2023.

[5] Yuming Zhang, Peng Wang, Ming Tan, and Wei-Guo Zhu. Learned adapters are better than manually designed adapters, 2023. URL https://api.semanticscholar.org/CorpusID:259858833.

[6] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A Survey of Large Language Models, March 2023.

[7] 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, 2021.

[8] Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment, 2023. URL https://api.semanticscholar.org/CorpusID:266362573.

[9] Ning Ding, Yujia Qin, Guang Yang, Fu Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Haitao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juan Li, and Maosong Sun. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models, 2022.

[10] Yi Xin, Siqi Luo, Haodi Zhou, Junlong Du, Xiaohong Liu, Yue Fan, Qing Li, and Yuntao Du. Parameter-efficient fine-tuning for pre-trained vision models: A survey, 2024. URL https://api.semanticscholar.org/CorpusID:267412110.

[11] Qingru Zhang, Minshuo Chen, Alexander W. Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning, 2023. URL https://api.semanticscholar.org/CorpusID:257631760.

[12] Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. Sparse low-rank adaptation of pre-trained language models, 2023. URL https://api.semanticscholar.org/CorpusID:265294736.

[13] Yahao Hu, Yifei Xie, Tianfeng Wang, Man Chen, and Zhisong Pan. Structure-aware low-rank adaptation for parameter-efficient fine-tuning, 2023. URL https://api.semanticscholar.org/CorpusID:264336659.

[14] Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. Autolora: Automatically tuning matrix ranks in low-rank adaptation based on meta learning, 2024.

[15] Zequan Liu, Jiawen Lyn, Wei Zhu, Xing Tian, and Yvette Graham. Alora: Allocating low-rank adaptation for fine-tuning large language models, 2024.

[16] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions, 2017.

[17] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search, 2019.

[18] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[19] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2016.

[20] Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Daniel M. Bikel, Lukas Blecher, Cristian Cantón Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony S. Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel M. Kloumann, A. V. Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, R. Subramanian, Xia Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zhengxu Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL `https://api.semanticscholar.org/CorpusID:259950998`.

[21] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one?, 2019.

[22] Qingru Zhang, Simiao Zuo, Chen Liang, Alexander Bukharin, Pengcheng He, Weizhu Chen, and Tuo Zhao. Platon: Pruning large transformer models with upper confidence bound of weight importance, 2022.

[23] Kaifeng Bi, Lingxi Xie, Xin Chen, Longhui Wei, and Qi Tian. Gold-nas: Gradual, one-level, differentiable, 2020.

[24] Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization, 2020.

[25] 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, 2018.

[26] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open misc question answering, 2018.

[27] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language, 2020.

[28] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions, 2019.

[29] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems, 2017.

[30] 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, 2021.

[31] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena, June 2023.

[32] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2020.

[33] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them, 2022.

[34] Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations, 2023.

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

[36] Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. The E2E dataset: New challenges for end-to-end generation, August 2017. URL https://aclanthology.org/W17-5525.

[37] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning, 2017. URL https://api.semanticscholar.org/CorpusID:25156106.

[38] Qidong Liu, Xian Wu, Xiangyu Zhao, Yuanshao Zhu, Derong Xu, Feng Tian, and Yefeng Zheng. Moelora: An moe-based parameter efficient fine-tuning method for multi-task medical applications, 2023.

[39] 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, 2024.

[40] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning, 2021.

[41] Xiao Liu, Kaixuan Ji, Yicheng Fu, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks, 2021.

[42] Wei Zhu, Aaron Xuxiang Tian, Congrui Yin, Yuan Ni, Xiaoling Wang, and Guotong Xie. Iapt: Instruction-aware prompt tuning for large language models, 2024.

[43] Elad Ben-Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models, 2021.

[44] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning, 2022. URL https://api.semanticscholar.org/CorpusID:248693283.

[45] Shengding Hu, Zhen Zhang, Ning Ding, Yadao Wang, Yasheng Wang, Zhiyuan Liu, and Maosong Sun. Sparse structure search for parameter-efficient tuning, 2022.

[46] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models, 2024.

[47] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[48] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback, 2022.

[49] Manas Gupta, Vishandi Rudy Keneta, Abhishek Vaidyanathan, Ritwik Kanodia, Efe Camci, Chuan-Sheng Foo, and Jie Lin. Global magnitude pruning with minimum threshold is all we need, 2024.

[50] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient Finetuning of Quantized LLMs, May 2023.

[51] Zikai Zhou, Qizheng Zhang, Hermann Kumbong, and Kunle Olukotun. Lowra: Accurate and efficient lora fine-tuning of llms under 2 bits, 2025.

[52] Shaoxiang Chen, Zequn Jie, and Lin Ma. Llava-mole: Sparse mixture of lora experts for mitigating data conflicts in instruction finetuning mllms, 2024.

[53] Shu Yang, Muhammad Asif Ali, Cheng-Long Wang, Lijie Hu, and Di Wang. Moral: Moe augmented lora for llms' lifelong learning, 2024.

[54] Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Jun Zhao, Wei Shen, Yuhao Zhou, Zhiheng Xi, Xiao Wang, Xiaoran Fan, et al. Loramoe: Revolutionizing mixture of experts for maintaining world knowledge in language model alignment, 2023.

[55] Yunhao Gou, Zhili Liu, Kai Chen, Lanqing Hong, Hang Xu, Aoxue Li, Dit-Yan Yeung, James T Kwok, and Yu Zhang. Mixture of cluster-conditional lora experts for vision-language instruction tuning, 2023.

[56] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017.

[57] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts, 1991.

[58] Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. Cross-task generalization via natural language crowdsourcing instructions, 2021. URL https://api.semanticscholar.org/CorpusID:237421373.

[59] Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners, 2021. URL https://api.semanticscholar.org/CorpusID:237416585.

[60] Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal V. Nayak, Debajyoti Datta, Jonathan D. Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng-Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Févry, Jason Alan Fries, Ryan Teehan, Stella Biderman, Leo Gao, Tali Bers, Thomas Wolf, and Alexander M. Rush. Multitask prompted training enables zero-shot task generalization, 2021. URL https://api.semanticscholar.org/CorpusID:239009562.

[61] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

[62] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing, 2020.

[63] OpenAI. GPT-4 Technical Report, March 2023.

[64] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement 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, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing, October 2020. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

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

[66] Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, Arvind Krishnamurthy University of Washington, and Duke University. Punica: Multi-tenant lora serving, 2023. URL https://api.semanticscholar.org/CorpusID:264590197.

# A. Appendix: more related works

Since LoRA is the most popular PEFT method in the era of large language models, many works are devoted to improving upon LoRA. There are three representative lines of works. The first line of works is to apply LoRA to the quantized LLM backbone [50]. These works are important since they allow the democratization of LLM usage. QLoRA proposes a novel quantization method for LLMs, and provide extensive experiments on fine-tuning the quantized LLMs with LoRA. LowRA [51] is introduced, the first framework to enable LoRA fine-tuning below 2 bits per parameter with minimal performance loss, and highlights the potential of ultra-low-bit LoRA fine-tuning for resource-constrained environments. The second line of research includes a series of works [38, 52–55] that looks into combining Mixture-of-Experts (MoE) [56, 57] and LoRA. MOELoRA [38] proves that fine-tuning LoRA modules with a MOE router enables the LLMs to perform well in a multi-task learning setting. LoRAMoE [54] integrates LoRAs using a router network to alleviate world knowledge forgetting after instruction tuning. MoCLE [55] proposes a MoE architecture to activate task-customized model parameters based on instruction clusters.

# B. Appendix for the datsets and evaluation metrics

## B.1. Commonsense reasoning tasks

**BoolQ** The BoolQ dataset, introduced by [28], is a benchmark dataset designed for training and evaluating models on the task of reading comprehension, specifically for answering yes/no questions. It comprises questions that are naturally occurring—sourced from real queries posed by people on various websites. Each question is paired with a corresponding passage from Wikipedia that provides the necessary context to answer the question. The dataset is notable for its diverse and challenging nature, featuring questions that require a deep understanding of the passage, inference, and sometimes common sense reasoning.

**OpenBookQA** The OpenBookQA [26] dataset is a benchmark designed to evaluate the ability of AI systems to understand and reason with elementary-level science knowledge. Created by the Allen Institute for AI, it includes multiple-choice questions, each with four possible answers. The questions are based on a core set of science facts that are typically found in a student's "open book" of basic science knowledge. Unlike straightforward fact-recall questions, OpenBookQA challenges models to apply, analyze, and reason about the facts, often requiring external common-sense knowledge to arrive at the correct answer.

**ARC** The AI2 Reasoning Challenge (ARC) dataset [25], developed by the Allen Institute for AI (AI2), is a benchmark for evaluating the ability of AI systems to perform complex reasoning over

science questions. The dataset is composed of science exam questions spanning multiple grade levels from third grade to ninth grade, collected from various sources such as textbooks, standardized tests, and other educational materials. The questions are divided into an Easy Set (ARC-e) and a Challenge Set (ARC-c), with the latter containing questions that require more sophisticated reasoning and understanding of scientific concepts.

**PIQA** The Physical Interaction Question Answering (PIQA) dataset [27] is designed to evaluate a model's understanding of physical interactions and common-sense reasoning. Developed by the Allen Institute for AI, PIQA consists of multiple-choice questions that focus on everyday scenarios and the practical use of objects. Each question presents a short description of a physical task and provides two possible solutions, challenging the model to select the most plausible one based on general physical knowledge and intuitive reasoning.

## B.2. Math reasoning tasks

**AQuA** The AQuA (Algebra Question Answering) dataset [29] is a comprehensive collection of algebraic problems designed to evaluate and enhance the problem-solving abilities of AI systems. It includes a wide range of questions covering various algebraic concepts, from basic arithmetic to more complex equations and word problems. Each problem is meticulously curated to test the system's ability to understand, interpret, and solve algebraic expressions and equations.

## B.3. Natural language understanding tasks

We experiment on three natural language understanding tasks from the GLUE [35] benchmark:

- The Stanford Sentiment Treebank (SST-2) is a widely used benchmark task in natural language processing (NLP) for binary sentiment classification. It consists of sentences extracted from movie reviews in the Rotten Tomatoes dataset, annotated to indicate whether the expressed sentiment is positive or negative. SST-2 simplifies the original Stanford Sentiment Treebank (SST), which included fine-grained sentiment labels, by focusing on a two-class classification problem. Each sentence is labeled at the phrase level, but the task is typically evaluated using sentence-level predictions.
- The Recognizing Textual Entailment (RTE) task, part of the GLUE (General Language Understanding Evaluation) benchmark, is a natural language inference challenge designed to evaluate a model's ability to determine whether a given hypothesis can be logically inferred (entailed) from a premise. Framed as a binary classification problem, RTE requires models to predict "entailment" (if the hypothesis necessarily follows from the premise) or "not entailment" (if it does not).
- The Question Natural Language Inference (QNLI) task, part of the General Language Understanding Evaluation (GLUE) benchmark, is designed to evaluate a model's ability to determine the relationship between a question and a given sentence. Adapted from the Stanford Question Answering Dataset (SQuAD), QNLI reformulates question answering as a binary classification problem. Each instance pairs a question with a sentence from a context passage, and the task is to predict whether the sentence contains the correct answer to the question (labeled as "entailment") or not ("not_entailment").

## B.4. Natural language generation tasks

We experiment on three natural language generation tasks:

- The E2E benchmark [36] dataset for training end-to-end, data-driven natural language generation systems in the restaurant domain. It asks a model to generate natural utterances based on a set of given key contents. This dataset has a 42061/4672/4693 train/dev/test split.

- WikiSQL [37] consists of a corpus of 87,726 hand-annotated SQL query and natural language question pairs. These SQL queries are further split into training (61,297 examples), development (9,145 examples) and test sets (17,284 examples). It can be used for natural language inference tasks related to relational databases. In this work, we will ask the LLMs to generate SQL queries based on the given natural language questions.

**GSM8k** The GSM8k dataset [30], also known as the Grade School Math 8k dataset, is a comprehensive collection designed for evaluating and training mathematical problem-solving abilities of machine learning models. Comprising 8,000 high-quality, diverse grade school math word problems, GSM8k serves as a benchmark for assessing the performance of models in understanding and solving arithmetic, algebraic, and logical reasoning challenges. Each problem in the dataset is meticulously curated to reflect real-world scenarios that students encounter in grade school, ensuring relevance and practicality.

## B.5. The MMLU benchmark

Massive Multitask Language Understanding (MMLU) [32] is a new benchmark designed to measure knowledge acquired during pretraining by evaluating large language models exclusively in zero-shot and few-shot settings. This makes the benchmark more challenging and more similar to how we evaluate humans. The benchmark covers 57 subjects across STEM, the humanities, the social sciences, and more. It ranges in difficulty from an elementary level to an advanced professional level, and it tests both world knowledge and problem solving ability. Subjects range from traditional areas, such as mathematics and history, to more specialized areas like law and ethics.

## B.6. The BBH benchmark

BIG-Bench Hard (BBH) [33] is a subset of the BIG-Bench, a diverse evaluation suite for language models. BBH focuses on a suite of 23 challenging tasks from BIG-Bench that were found to be beyond the capabilities of current language models. These tasks are ones where prior language model evaluations did not outperform the average human-rater.

## B.7. The MT-Bench dataset

The MT-Bench [31] dataset is a widely used dataset for evaluating the quality of LLMs. It contains 80 questions. The LLMs generate responses for these questions, and human annotators or LLM annotators will judge the quality of these responses.

## B.8. Instruction tuning datasets

Instruction tuning is an important method to improve the general capabilities of large language models [48]. With the rise of large language models in the scale of 10B parameters or more, like GPT-3, T5, PaLM, researchers have actively explored the few-shot or zero-shot capabilities of these models. [58] find that fine-tuning these LLMs on a large scale datasets containing hundreds of NLP tasks significantly improves the zero-shot performances on unseen tasks, establishing the scaling law of task numbers. The previous works like [59] and T0 [60] establishes the instruction tuning datasets by transforming the traditional NLP tasks into a unified prompt format. Instruct-GPT [48] conducts instruction tuning using the dataset constructed based the user queries from the OpenAI API users. Note that this work is also a seminal work for human feedback learning with reinforcement learning. However, the complete instruction tuning dataset from [48] remains closed. With the launch of ChatGPT, [61] (Alpaca) constructs an instruction tuning dataset with diverse topics using the self-instruct techniques.

For our experiment, we employ two general-purpose instruction tuning datasets:

- The UltraChat dataset [34] is a large-scale, synthetically generated conversational corpus designed to train and enhance AI-driven dialogue systems. Comprising over 1.5 million

multi-turn dialogues and 56 million conversation turns, it leverages OpenAI's GPT-3.5-turbo to simulate diverse, human-like interactions across a wide array of topics, including daily life, technical discussions, and creative role-playing scenarios. Structured as user-assistant exchanges, the dataset emphasizes naturalness, coherence, and contextual depth, supporting the development of robust language models capable of handling nuanced conversations. Available in English, Chinese, and Japanese, UltraChat serves as a versatile resource for fine-tuning large language models (LLMs), advancing research in dialogue systems, and improving cross-lingual applications. Its synthetic nature allows scalability while maintaining high-quality, varied interactions, making it accessible via platforms like Hugging Face for researchers and developers aiming to push the boundaries of conversational AI.

- Alpaca dataset [61]. Specifically, we employs its cleaned version[7]. This dataset comprises 51K instructions and demonstrations, and is suitable for instruction tuning. The cleaned version corrects multiple issues such as hallucinations, merged instructions, and empty outputs.

The detailed statistics of the above tasks' datasets are presented in Table 5.

| Datasets | #train | #dev | #test | Type | Metrics |
|---|---|---|---|---|---|
| ***Commonsense reasoning tasks*** | | | | | |
| BoolQ | 9427 | - | 3270 | Commonsense reasoning | acc |
| OBQA | 4957 | 500 | 500 | Commonsense reasoning | acc |
| ARC-e | 2251 | 570 | 2376 | Commonsense reasoning | acc |
| ARC-c | 1119 | 299 | 1172 | Commonsense reasoning | acc |
| PIQA | 16,000 | 2,000 | 3,000 | Commonsense reasoning | acc |
| ***Math reasoning tasks*** | | | | | |
| AQuA | 97467 | 254 | 254 | Math reasoning | acc |
| GSM8K | 7473 | - | 1319 | Math reasoning | acc |
| ***Natural language understanding tasks*** | | | | | |
| SST-2 | 66k | 1k | 0.8k | Sentiment classification | acc |
| RTE | 2.5k | 0.1k | 0.1k | Natural language inference | acc |
| QNLI | 104k | 1k | 5.4k | Natural language inference | acc |
| E2E | 42k | 4.6k | 4.6k | Natural language generation | rouge |
| WikiSQL | 61k | 9K | 17K | SQL generation | acc |
| ***Instruction tuning*** | | | | | |
| UltraChat | 56M | - | - | Instruction tuning | - |
| Alpaca | 50k | - | - | Instruction tuning | - |
| ***LLM evaluation tasks*** | | | | | |
| MT-Bench | - | - | 80 | Question answering | GPT-4 scores |
| MMLU | - | - | 14042 | Question Answering | acc |
| BBH | - | - | 6,511 | Question Answering | acc |

Table 5: The dataset statistics.

## B.9. Evaluation metrics/protocols

For the commonsense reasoning and math reasoning tasks, since they usually come with a definite answer choice, we will directly consider the correctness of the final answers. Thus, we report accuracy (denoted as acc).

For evaluating the quality of instruction tuned LlaMA-2 7B on the MT-Bench, we follow the current common practice of utilizing GPT-4 as a unbiased reviewer [31]. We generate model responses from a fine-tuned model with beam size 3 with the generation function in Huggingface Transformers [62]. Then we compare MOELoRA and ShapLoRA's answers with GPT-4. For each instruction in MT-Bench, GPT-4 [63] is asked to write a review for both answers from the two methods, and

[7] `https://huggingface.co/datasets/yahma/alpaca-cleaned`.

assigns a quantitative score on a scale of 10 to each response. The prompts of instructing GPT-4 for evaluation is presented in Appendix C.

# C. Prompt templates for GPT-4 evaluations

In this work, we utilize the powerful LLM GPT-4 [63] as the evaluator for comparing the instruction tuning quality. As a reviewer, GPT-4 will receive a query [query], two responses, [response1] and [response2], from two assistants. We will ask GPT-4 to write a review for each response, assessing the quality of the response, and then ask GPT-4 to assign a score on a scale of 10 to each response.

Template for prompt:

```
Task Introduction
you will be given a query, and two responses
from two assistants,
could you compare the two responses,
and do the following:
(1) write a concise review for each
assistant's response, on how well the
response answers the query, and whether
it will be helpful to humans users, and any
issues in the response;
(2) assigns a quantitative score on a scale
of 10 to each response, reflecting
your assessment of the two responses
Query:
[query]
Response 1 from assistant 1:
[response1]
Response 2 from assistant 2:
[response2]
```

# D. Appendix for Experimental settings

Here, we provide more details for experimental settings.

**Hyper-parameters for the baseline PEFT methods** For P-tuning V2, the number of prompt tokens at each layer is set to 160. For IAPT, the prompt's length is set to 8, the bottleneck dimension is set to 256, and the number of prompt layers is set to 32. For adapter-based methods, the bottleneck dimension is set to 40, and the adapter modules are added on the self-attention and feed-forward module. For LoRA and ALoRA, the initial rank at each module is set to 8. For AdaLoRA and AutoLoRA, the initial rank at each module is set to 16, and half of the rank budget is pruned during fine-tuning. We adjust the sparsity for SSP so that the number of tunable parameters is comparable with ShapLoRA and the other baselines. For BitFit, the bias vectors are first initialized with 16 dimensions, and then are projected to the dimensions that are aligned with the linear modules of the Transformer backbone. For $(IA)^3$, the product vectors are first initialized with 32 dimensions, and then are projected to the dimensions that are aligned with the linear modules of the Transformer backbone.

**Training settings for PEFT methods** We use the HugginFace Transformers [64] and PEFT [65] for implementing all the methods, and for training and making predictions. For fine-tuning LlaMA-3 8B model, the maximum sequence length is set to 2048. The maximum training epoch is set to 10. The batch size is set between 16 for task with less than 10k training set, and 128 otherwise. We use AdamW as the optimizer with a linear learning rate decay schedule and 6% of the training steps for warm-up. The learning rate is set to 1e-4. The other hyper-parameters are kept the same with [64].

In every 200 steps, the model is evaluated on the dev set. Patience is set to 10, that is, if the model does not achieve a lower development set loss for 10 evaluation runs, the training stops. The best checkpoint on the dev set is used to run predictions on the test set.

# E. Appendix: more experimental results

**Results on the NLP and NLG tasks** Table 6 reports the experimental results on the SST-2, RTE, QNLI, E2E, and WikiSQL tasks.

| Method | SST-2 (acc) | RTE (acc) | QNLI (acc) | E2E (rouge) | WikiSQL (acc) |
|---|---|---|---|---|---|
| ***Baselines*** | | | | | |
| Housbly-Adapter | 94.3 | 86.7 | 93.1 | 72.2 | 86.8 |
| BitFit | 94.5 | 86.9 | 93.7 | 72.7 | 87.3 |
| P-tuning v2 | 93.8 | 84.5 | 92.2 | 71.3 | 86.4 |
| LoRA | 95.2 | 87.2 | 94.1 | 73.5 | 87.3 |
| AdaLoRA | 95.2 | 87.3 | 94.0 | 73.8 | 86.9 |
| AutoLoRA | 95.5 | 87.6 | 93.9 | 73.6 | 87.5 |
| ***Our proposed methods*** | | | | | |
| ShapLoRA | **96.1** | **89.0** | **95.1** | **74.8** | **88.5** |

Table 6: The Overall comparison of the three GLUE tasks and two natural language generation tasks. The backbone model is LlaMA-3 8B. We report the median performance over five random seeds. The metric for each task is explained in Appendix B.9.

**Results on different LLM backbones** Table 7 reports the experimental results with different LLM backbones.

| Method | BoolQ (acc) | PIQA (acc) | MMLU (acc) |
|---|---|---|---|
| ***Results for Deepseek distilled Llama-3 8B*** | | | |
| MOELoRA | 83.2 | 85.8 | 59.4 |
| ShapLoRA | **84.1** | **86.4** | **60.2** |
| ***Results for Deepseek distilled Qwen2.5 3B*** | | | |
| MOELoRA | 78.8 | 82.3 | 56.2 |
| ShapLoRA | **80.7** | **83.5** | **57.1** |

Table 7: Results for different PEFT methods on the BoolQ, PIQA and MMLU benchmarks. The backbone LLMs are the distilled Llama-3 8B and Qwen2.5 3B from Deepseek R1 [47].

# F. Appendix: inference efficiency

To demonstrate the inference efficiency of our ShapLoRA method, we now compare the GPU memory and decoding speed of ShapLoRA, AutoLoRA, and MOELoRA under beam search with different beam sizes. We use the test sets of the experimented tasks for efficiency evaluation. In this experiment, LoRA parameters are not merged to the backbone to mimic the single-LLM multi-LoRA setting [66]. We present two metrics for measuring efficiency: (a) peak memory cost (in GiB). (b) tokens generated per second (tps). The results are presented in Table 8. From Table 8, under beam sizes 1 and 3, the ShapLoRA method has a comparable decoding speed and memory cost with the other baselines.

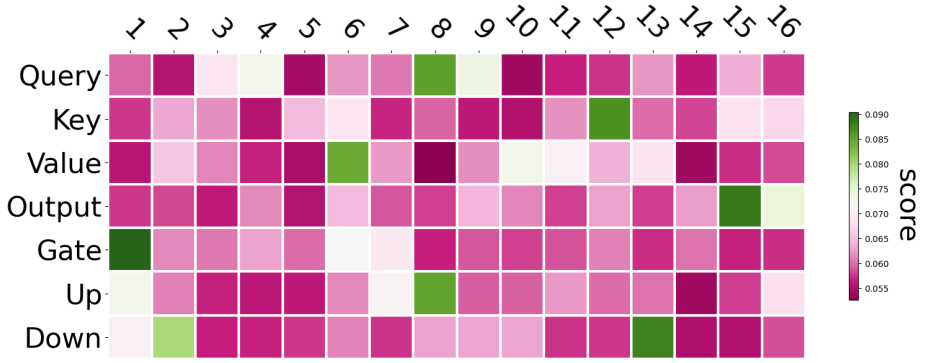
# G. Appendix: Visualization of Shapley sensitivity importance scores

In Figure 3 and 4, we present the LoRA importance scores on LlaMA-3 8B on the BoolQ and PIQA tasks.

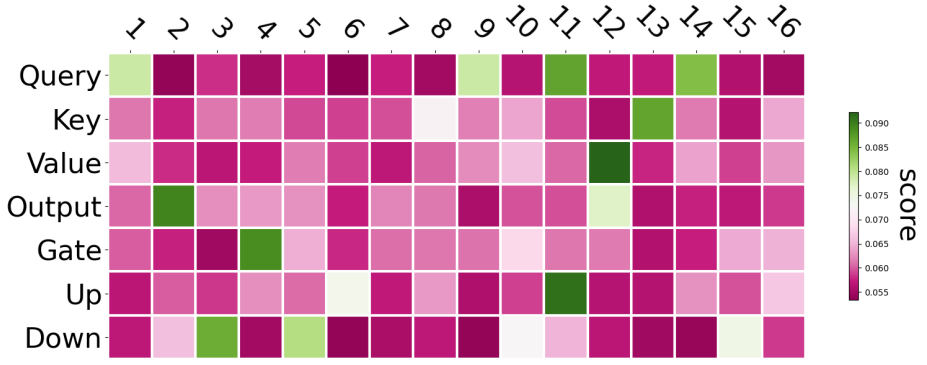
| Method | Beam size | Speed (tps) | Memory cost (MiB) |
|---|---|---|---|
| AutoLoRA | 1 | 36.7 | 13.8 |
| | 3 | 30.4 | 15.4 |
| MOELoRA | 1 | 32.5 | 13.8 |
| | 3 | 26.4 | 15.6 |
| ShapLoRA | 1 | 37.1 | 13.7 |
| | 3 | 30.7 | 15.2 |

Table 8: The memory and speed of different PEFT methods during inference.

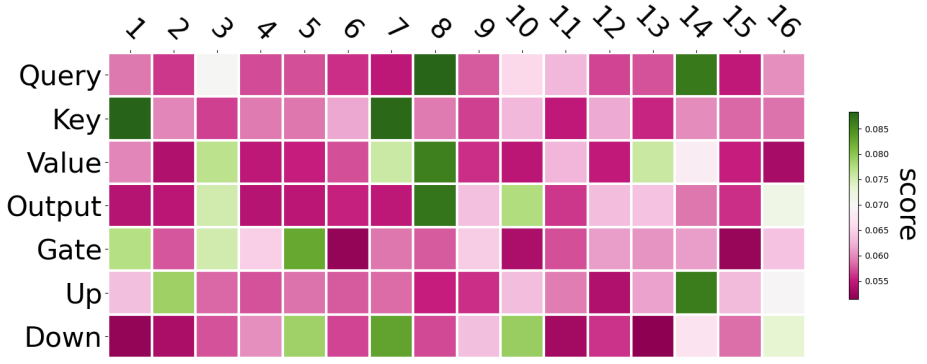
In Figure 5, we present our ShapLoRA's rank allocation results when fine-tuning LlaMA-3 8B on the BoolQ and PIQA tasks.



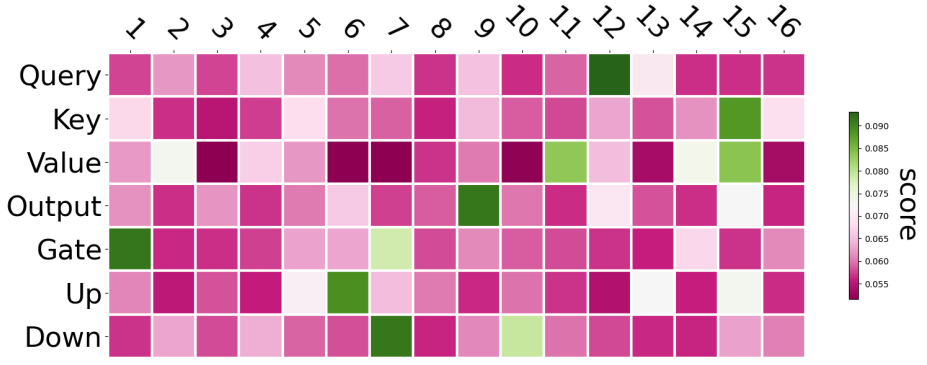

(a) 8th layer




(b) 16th layer

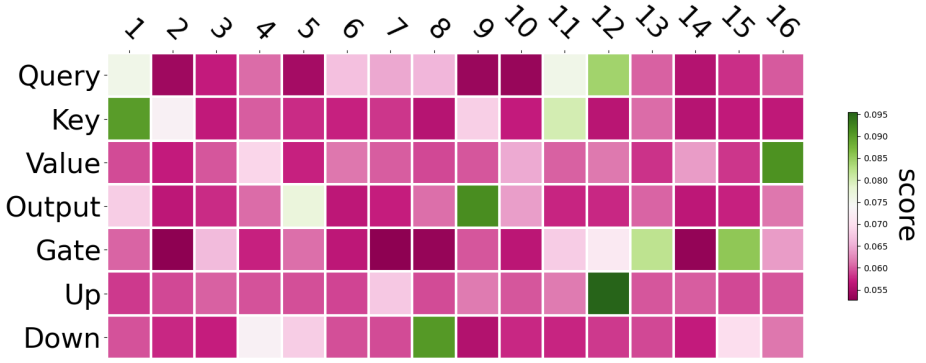



(c) 24th layer

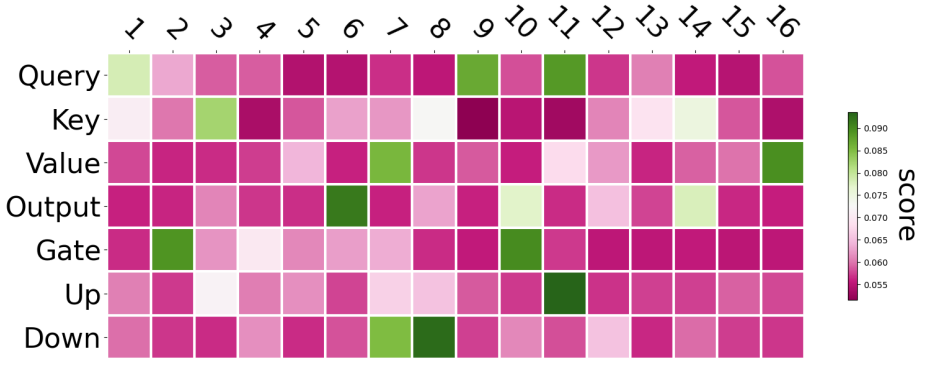



(d) 32nd layer

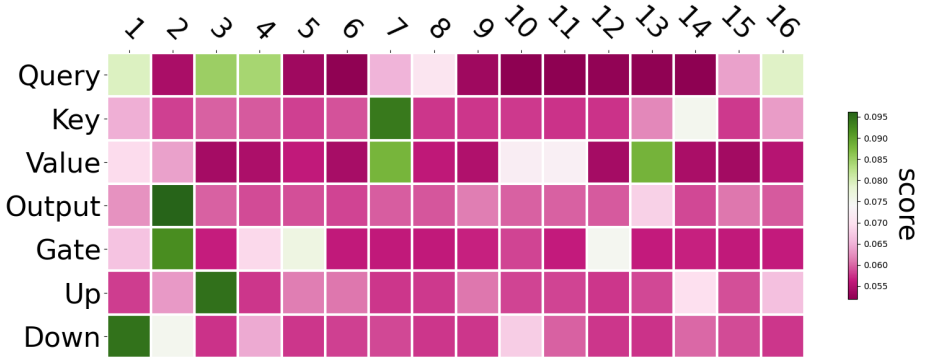
Figure 3: Shapley sensitivity importance scores of LoRA ranks on the 8th, 16th, 24th, and 32nd layers of LLaMA-3 8B after finetuning on BoolQ.



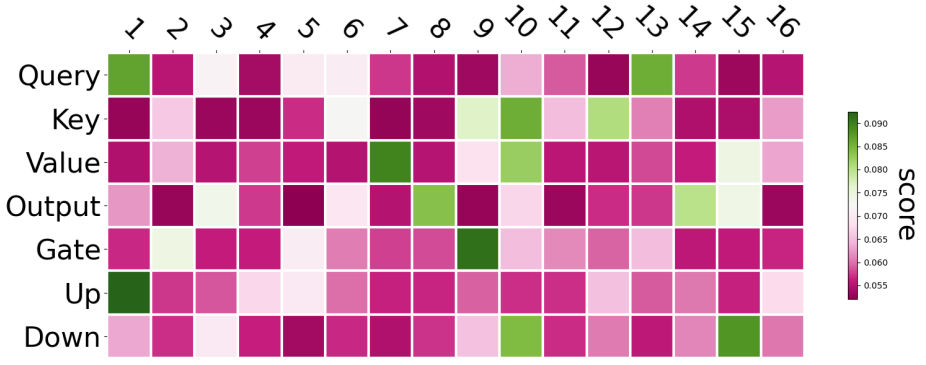

(a) 8th layer




(b) 16th layer

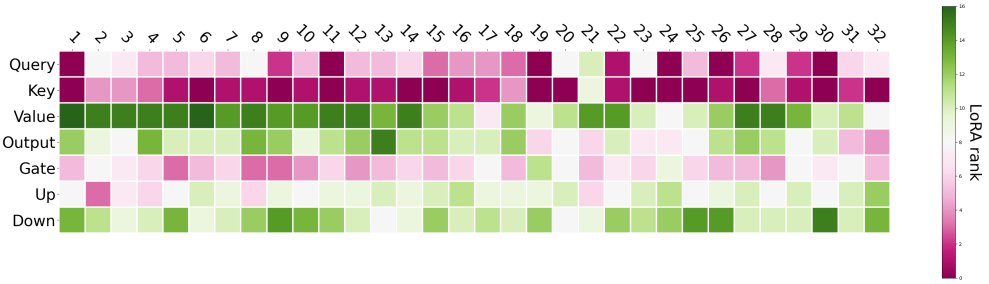



(c) 24th layer

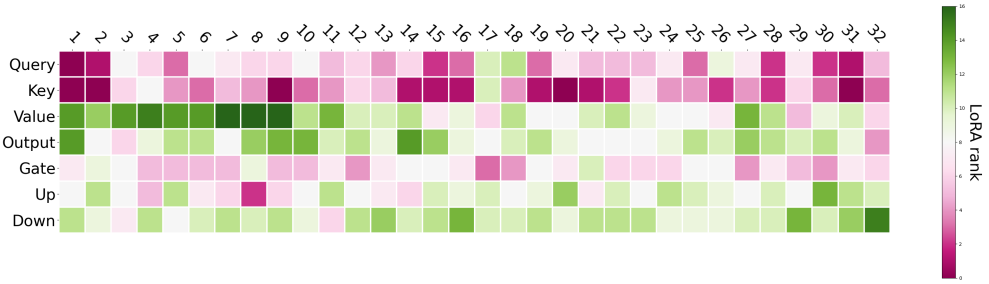



(d) 32nd layer

Figure 4: Shapley sensitivity importance scores of LoRA ranks on the 8th, 16th, 24th, and 32nd layers of LLaMA-3 8B after finetuning on PIQA.




(a) BoolQ




(b) PIQA

Figure 5: ShapLoRA's LoRA rank allocation results on the LLaMA-3 8B backbone after finetuning on BoolQ and PIQA.