# PEFT-Bench: A Parameter-Efficient Fine-Tuning Methods Benchmark

**Robert Belanec**[♠†], **Branislav Pecher**[†], **Ivan Srba**[†], **Maria Bielikova**[†]
[♠] Faculty of Information Technology, Brno University of Technology, Brno, Czechia
[†] Kempelen Institute of Intelligent Technologies, Bratislava, Slovakia
{name.surname}@kinit.sk

## Abstract

Despite the state-of-the-art performance of Large Language Models (LLMs) achieved on many tasks, their massive scale often leads to high computational and environmental costs, limiting their accessibility. Parameter-Efficient Fine-Tuning (PEFT) methods address this challenge by reducing the number of trainable parameters while maintaining strong downstream performance. Despite the advances in PEFT methods, current evaluations remain limited (in terms of evaluated models and datasets) and difficult to reproduce. To bridge this gap, we introduce *PEFT-Bench*, a unified end-to-end benchmark for evaluating diverse PEFT methods on autoregressive LLMs. We demonstrate its usage across *27 NLP datasets* and *7 PEFT methods*. To account for different PEFT training and inference factors, we also introduce the *PEFT Soft Cost Penalties (PSCP)* metric, which takes trainable parameters, inference speed, and training memory usage into account.

## 1 Introduction

Large Language Models (LLMs) (Radford et al., 2019; Achiam et al., 2023; Dubey et al., 2024; Jiang et al., 2023; Team et al., 2024) achieve remarkable results in many Natural Language Understanding (NLU) and Natural Language Generation (NLG) tasks. This significant increase in performance was possible due to the introduction of the attention mechanism forming the transformer architecture (Vaswani, 2017). To achieve such performance, many currently best-performing models contain a vast number of attention blocks, increasing the number of trainable parameters to millions (Devlin et al., 2019), billions (Radford et al., 2019), or even trillions (Fedus et al., 2022). With the increase in parameters, the computational and data requirements for training also increased, making the development of LLMs unavailable for many academic institutions or practitioners, as well as producing a bigger carbon footprint. To address this issue, a new paradigm was introduced, denoted as "pre-train, fine-tune", in which the language model is at first pre-trained on large amounts of raw data and then, in a fully supervised way, adapted to the downstream task (Liu et al., 2023a). This standard fine-tuning still requires substantial amounts of resources as well as significant quantities of data and storage to store fine-tuned weights for each task.

Parameter-Efficient Fine-Tuning (PEFT) methods aim to tackle these problems by reducing the number of trainable parameters while maintaining the performance on a set of downstream tasks. Over the last years, we have witnessed a steady increase in new PEFT methods being introduced in research works (Prottasha et al., 2025). However, in practice, due to their efficiency and popularity, predominant LoRA-based methods are commonly employed with autoregressive LLMs, and other PEFT methods are rarely chosen. Moreover, a unified and fully open-sourced evaluation of PEFT methods of different types is still missing. We can summarize the cause within the following points: 1) When using the existing PEFT methods as baselines, the lack of fully functional open-source implementations and essential details on the experimental setup (Asai et al., 2022; Shi and Lipani, 2024; Tang et al., 2025) often prevents fellow researchers from replicating their results. 2) Evaluating only non-autoregressive models often restricts the generalizability of the experiments and makes the method look obsolete in the current era of autoregressive LLMs. 3) Finally, the existing PEFT methods are often evaluated primarily on NLU tasks (especially GLUE and SuperGLUE tasks).

To ultimately improve and unify the evaluation of PEFT methods, we propose a new benchmark **PEFT-Bench**[1], which bridges the gap in the com-

[1]To promote the replicability and usage of our benchmark, we provide a dedicated GitHub repository: https://github.com/kinit-sk/PEFT-Bench.



*Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics*
*Volume 1: Long Papers*, pages 3035–3054

parison of PEFT methods. We simultaneously introduce the **PEFT-Factory framework**[2] (Belanec et al., 2025b), which provides a necessary underlying technological support for execution of the PEFT-Bench benchmark.

Our main contributions are as follows:

- We introduce the *PEFT-Bench*, an end-to-end benchmark that defines the datasets, metrics, and methodology for evaluating PEFT methods in NLP in a fair and consistent environment. To the best of our knowledge, we are the first to provide such a benchmark in NLU and NLG tasks for PEFT methods with autoregressive LLMs. To demonstrate its use, we evaluate *7 different PEFT methods* in terms of efficiency and stability with limited data on *27 NLP datasets*.
- To unify and automate the evaluation, we simultaneously introduce the *PEFT-Factory* framework. The framework allows researchers to extend PEFT-Bench with new PEFT methods, maintaining the same evaluation and comparison setup.
- Finally, *we propose the PEFT Soft Cost Penalties (PSCP) metric*, which introduces a number of trainable parameters, memory usage, and inference speed in the final score calculation. PSCP reflects the practical feasibility of deploying PEFT methods in real-world scenarios, where efficiency and resource constraints are often as critical as task performance.

## 2 Related Work

Many new PEFT methods are primarily evaluated on GLUE and SuperGLUE benchmarks and often include the main results only on non-autoregressive models (Tang et al., 2025; Zhang et al., 2025b,a) (e.g., T5 (Raffel et al., 2020), RoBERTa (Liu et al., 2019)) or include only limited evaluation on autoregressive models (Lan et al., 2025; Shi and Lipani, 2024; Belanec et al., 2025a). Although some of the new parametrization-based methods (e.g., SVFT (Lingam et al., 2024), SVF (Sun et al., 2025)) are already evaluated on autoregressive models and on natural language generation tasks, a more unified and overarching evaluation is still missing.

While several survey papers on PEFT methods include performance evaluations (Ding et al., 2023; Lialin et al., 2023; Xu et al., 2023), these are largely limited to encoder-based or encoder–decoder-based models and do not consider autoregressive models (similarly to the evaluation of individual new PEFT methods mentioned in the previous paragraph). A notable exception is the most recent survey (Prottasha et al., 2025), which does include autoregressive models, but only for a small set of NLG tasks.

Prior benchmarks have primarily focused on evaluating PEFT methods in the computer vision domain (Zhang et al., 2023; Xin et al., 2024), leaving benchmarking on textual datasets underexplored. Moreover, such benchmarks often keep their implementation either fully or partially closed-sourced and do not provide a usable interface to rerun them with newly created PEFT methods.

Additionally, due to low baseline replicability, we have been witnessing a trend in new soft prompt-based methods to draw numerical results directly from the related works (without actually rerunning the experiments) (Zhang et al., 2025b; Tang et al., 2025). This practice is often not only methodologically incorrect (if the authors do not perfectly match the original environment), but also weakens the results of the proposed method.

In our work, we aim to mitigate these problems. Our PEFT-Bench is a fully open-source benchmarking approach for PEFT methods with state-of-the-art autoregressive NLP models, and with an easy-to-use interface for fellow researchers to add new methods. In addition, our work is the first systematic end-to-end approach (defining the datasets, metrics, and methodology) to provide a unified evaluation of PEFT methods for NLP.

## 3 PEFT-Bench

Our PEFT-Bench is constructed in multiple steps, which are visualised in the diagram in Figure 1. At a high level, it consists of three main components: 1) a diverse collection of datasets and tasks; 2) representative language models together with a set of PEFT methods; and 3) a set of evaluation metrics.

### 3.1 Tasks and Datasets

The PEFT-Bench includes **27 datasets** representing **12 unique tasks** in NLU and NLG. These datasets are categorized into 3 main groups: 1) NLU and Reasoning (further subcategorized into GLUE, SuperGLUE, and Others); 2) Math; and 3) Code Generation. An overview of the datasets contained in PEFT-Bench can be seen in Figure 2 (detailed

[2]PEFT-Factory source code is available at: https://github.com/kinit-sk/PEFT-Factory.

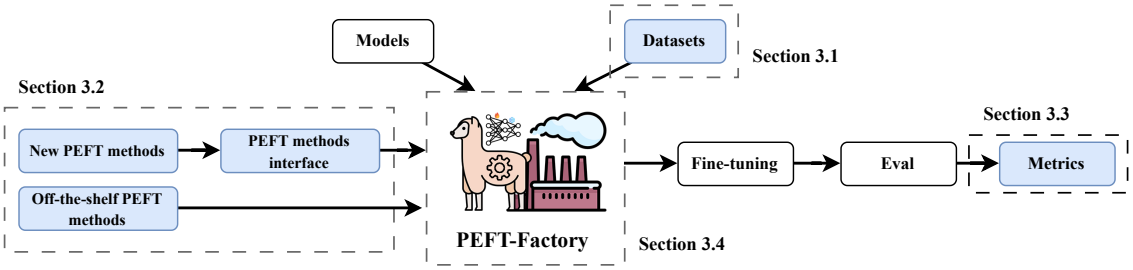


Figure 1: Diagram describing the methodology of **PEFT-Bench**. Blue components represent our contributions. We design **PEFT-Factory**, a framework based on LLaMA-Factory (Zheng et al., 2024) backbone to implement off-the-shelf methods from the HuggingFace PEFT library and an easy-to-use interface for new PEFT methods. Using these methods, we train LLaMa on selected datasets, which we have also included in the backbone. After training, we evaluate and compute the metrics for each model, method, and dataset combination.

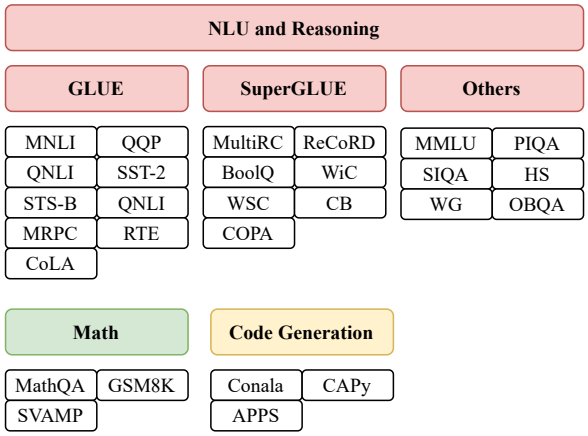


Figure 2: A diagram showing the overview and categorizations of datasets used in PEFT-Bench, totaling 27 datasets categorized into 3 main groups – NLU and Reasoning, Math, and Code Generation.

information about datasets can be found in the Appendix A, Table 4). We chose these datasets based on the following criteria: 1) The datasets represent standard tasks, commonly used for fine-tuning in the NLP domain. 2) The datasets provide high task and domain diversity. 3) The size and sequence lengths of a dataset are not overly large (training requires reasonable computational resources).

Current works on new PEFT methods (Asai et al., 2022; Shi and Lipani, 2024; Tang et al., 2025) often evaluate their performance on GLUE (Wang et al., 2018) and SuperGLUE (Wang et al., 2019) benchmarks. However, since GLUE and SuperGLUE became standard for the evaluation of NLU, they are often used in LLM pre-training and aligning many state-of-the-art models. Due to this, we are also including tasks that are newer and generally harder for LLMs, like code generation or math problem solving. Since we are using supervised fine-tuning to fine-tune an instruction-fine-tuned model, we also include an instruction in each sample (our defined templates of these instructions for each dataset can be found in Appendix A.2).

#### 3.1.1 Natural Language Understanding and Reasoning Tasks

Natural language understanding (NLU) is an area of tasks that aim to measure how language models perform in common linguistic problems (e.g., sentiment classification or coreference resolution). Evaluating such tasks became standard when designing a new method or training a new model. This area of tasks is represented by popular benchmarks GLUE and SuperGLUE, which we decided to include in PEFT-Bench.

Reasoning tasks can be either formed as reasoning understanding or as reasoning generation. In our task selection, we include both such forms of reasoning (i.e., reasoning classification and mathematical reasoning generation).

**GLUE Benchmark.** PEFT-Bench includes 8 datasets from GLUE, split into 5 tasks, namely **natural language inference (NLI)** – *MNLI* (Williams et al., 2018), *QNLI* (Rajpurkar et al., 2016), *RTE* (Dagan et al., 2005; Bar Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009); **paraphrase classification** – *QQP* [3], *MRPC* (Dolan and Brockett, 2005); **sentiment classification** – *SST-2* (Socher et al., 2013); **sentence similarity** – *STS-B* (Cer et al., 2017) and **acceptability classification** – *CoLA* (Warstadt et al., 2019).

**SuperGLUE Benchmark.** Additionally, PEFT-Bench also includes 7 datasets from SuperGLUE,

[3] https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

split into 4 tasks, namely **natural language inference (NLI)** – *CB* (De Marneffe et al., 2019); **question answering** – *MultiRC* (Khashabi et al., 2018), *ReCoRD* (Zhang et al., 2018), *BoolQ* (Clark et al., 2019), *COPA* (Roemmele et al., 2011); **word sense disambiguation** – *WiC* (Pilehvar and Camacho-Collados, 2019) and **coreference resolution** – *WSC* (Levesque et al., 2011);

All of the GLUE and SuperGLUE datasets are designed as classification problems with the exception of STS-B (which is a regression problem) and ReCoRD (which includes only open questions). To be able to train an autoregressive model that generates text on such classification tasks, we replace numerical class representations with textual labels (e.g., in the QQP dataset, we replace 0 with not_duplicate and 1 with duplicate).

**Other datasets.** PEFT-Bench also includes datasets that are, to the best of our knowledge, not part of any well-known benchmarks or are benchmarks on its own, but are still popular and commonly used to evaluate NLU and reasoning in LLMs, totaling 6 datasets split into 3 tasks, namely **question answering** – *MMLU* (Hendrycks et al., 2021), *PIQA* (Bisk et al., 2020), *SIQA* (Sap et al., 2019), *OBQA* (Khot et al., 2019); **natural language inference (NLI)** – *HellaSwag* (Zellers et al., 2019); **commonsense reasoning** – *WinoGrande* (Sakaguchi et al., 2021);

#### 3.1.2 Math Tasks

Another popular task that is used for LLM evaluation is mathematical problem-solving. The difficulty of datasets for such evaluation can range from simple question answering to full mathematical reasoning and problem-solving. PEFT-Bench includes 3 datasets for mathematical problem solving split into 3 tasks, namely **question answering** – *MathQA* (Amini et al., 2019); **math word problems** – *GSM8K* (Cobbe et al., 2021) and **simple math problems** – *SVAMP* (Patel et al., 2021).

#### 3.1.3 Code Generation Tasks

Finally, code generation is another popular task that also contributes to the domain diversity of PEFT-Bench, since it contains mostly code generation questions, which are out-of-domain for the previously mentioned tasks. We mostly focus our selection on the Python language. PEFT-Bench includes 3 datasets for code generation, namely *Conala* (Yin et al., 2018), *CodeAlpacaPy* (Chaudhary, 2023), and *APPS* (Hendrycks et al., 2021).

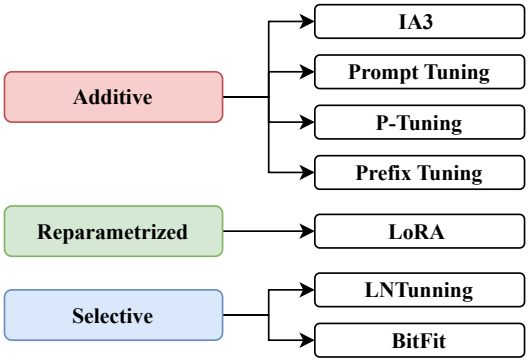


Figure 3: We evaluate methods from additive, reparametrized, and selective PEFT categories. The diagram shows the categorization of each method.

### 3.2 PEFT Methods and Pretrained Models

With the popularization of PEFT, many new methods are being introduced within a short period of time (Xu et al., 2023; Prottasha et al., 2025). Therefore, it is computationally expensive to evaluate them all. We design our PEFT method selection in PEFT-Bench to cover *additive PEFT*, *reparametrized PEFT*, and *selective PEFT* categories (the categorization of methods that we have selected can be found in Figure 3). To account for each category, we select **IA**$^3$ (Liu et al., 2022a), **Prompt Tuning** (Lester et al., 2021), **P-Tuning** (Liu et al., 2022b), and **Prefix Tuning** (Li and Liang, 2021) from additive PEFT methods, where IA$^3$ is an adapter-based method, and the rest are soft prompt-based methods. Consequently, we select **LoRA** (Hu et al., 2022) from reparametrized PEFT methods and **LNTuning** (Zhao et al., 2024) and **BitFit** (Ben Zaken et al., 2022) from selective PEFT methods. These methods are widely adopted, frequently used as baselines, and together represent one of the most influential and diverse approaches within each PEFT category. We provide a more detailed description of each selected PEFT method in Appendix C.

Since the efficiency of PEFT strongly depends on the base model architecture, we select a popular and open-source representative foundation model, **LLaMa-3-8B-Instruct** (Dubey et al., 2024), to evaluate all the mentioned PEFT methods. This model also represents a compromise in terms of the number of parameters and performance (we want to train a large enough model to fit within our computational resources).

### 3.3 Evaluation metrics

To measure task **performance** in PEFT-Bench, for each dataset, we use standard metrics, namely, F1 for classification tasks and multi-choice QA, accuracy for open answer QA and math problem-solving, and CodeBLEU (Ren et al., 2020) for code generation problems. CodeBLEU is an alternative to the standard BLEU metric that also evaluates syntax for different programming languages.

Current works on PEFT methods often measure solely the accuracy performance or the **number of trainable parameters**, which makes it harder to compare. Moreover, related works only rarely report the **memory and inference efficiency** in their results. These are the key factors that may affect the preference for using a specific method over another within a certain computational budget. At the same time, for efficient comparison of benchmarked methods, we need to aggregate performance and such additional PEFT training and inference factors into a single metric. To this end, we propose a new PEFT-bencharking metric called **PEFT Soft Cost Penalties (PSCP)**.

We propose to penalize the performance when the costs behind a certain factor are too high. Therefore, we need to scale the performance on the specific task $P_t$ with a penalty for that factor, similarly to the Cobb-Douglass production function (Cobb and Douglas, 1928) for $n$ costs:

$$y = P_t \times \prod_{i=1}^{n} cost_i^{\beta_i}, \tag{1}$$

where $\beta_i$ is the importance of $cost_i$, and $cost_i^{\beta_i}$ is the final penalty.

In our case, we consider the following factors: 1) the **number of trainable parameters** $M_p \in \mathbb{N}$; 2) **inference speed** $M_f \in \mathbb{R}^+$ (i.e., floating point operations); and 3) average **memory usage during training** $M_m \in \mathbb{R}^+$. Since, these factors are not normalized and cannot be used as penalties on their own, we normalize $M_p$, $M_f$, $M_m$ using reference constants $\frac{M_p}{C_p}$, $\frac{M_f}{C_f}$, $\frac{M_m}{C_m}$, where $C_p \in \mathbb{N}$, $C_f \in \mathbb{R}^+$ and $C_m \in \mathbb{R}^+$. The constants can be set either to a large value that we would not like to achieve (e.g., number of parameters close to full model training, double the inference floating point operations, or maximum memory available for training), or alternatively, they can also be set to the median or geometric mean of all available $M$ values from a specific factor. If we plug this normalization into Equation 1, we would end up with the intermediate equation 2.

$$y = P_t \times (\frac{M_p}{C_p})^{\beta_p} \times (\frac{M_f}{C_f})^{\beta_f} \times (\frac{M_m}{C_m})^{\beta_m} \tag{2}$$

This, however, would rescale the performance $P_t$ to infinity if $M >> C$, therefore we need to bound it to the interval $[0, 1)$ with the reciprocal function (i.e., $x^{-1}$). For this reason, we also need to add 1, so that the result of the reciprocal does not go to infinity when $M << C$. The final form of our PSCP metric is given by Equation 3.

$$PSCP = P_t \times (1 + \tfrac{M_p}{C_p})^{-\beta_p} \times (1 + \tfrac{M_f}{C_f})^{-\beta_f} \times (1 + \tfrac{M_m}{C_m})^{-\beta_m} \tag{3}$$

The proposed metric is easily extensible if additional factors are of interest. In our case, we consider factors that we want to minimize (i.e., the number of parameters, inference speed, and training memory). If we want to include different factors that we want to maximize (i.e., higher is better), we need to switch the numerator and denominator of the fraction for that cost calculation.

As the PSCP allows setting the importance of each cost by setting the $\beta$ value, it makes the PSCP metric further configurable and adaptable for different use cases. If we set the importance of the cost $\beta$ to 0, the resulting penalty will be 1, leaving the final PSCP metric unaffected by that factor. If we set the importance of the cost $\beta$ to a number greater than 0, the resulting penalty for that factor will increase (i.e., $cost_i^{\beta_i}$ will get closer to 0).

Subsequently, if the importance of all factors is set equally, only the magnitude of the PSCP score changes, not the order of methods. However, this occurs only when the performance differences between the compared methods are high. This can be seen in Appendix B in Table 9. If we want to put more emphasis on a particular cost, we need to set its $\beta$ value greater than that of other costs. This specific behaviour can be seen in Appendix B and Table 10a, where we decrease the importance of the number of trainable parameters and, therefore, decrease the penalty, which makes LoRA the method with the highest PSCP score.

### 3.4 PEFT-Factory

To increase the replicability, modularity, and usability of the PEFT-Bench, we simultaneously introduce **PEFT-Factory** (Belanec et al., 2025b),

a framework for efficient training of autoregressive LLMs, based on one of the popular LLaMA-Factory LLM training tools (Zheng et al., 2024). PEFT-Factory includes new off-the-shelf PEFT methods from the HuggingFace PEFT library (Mangrulkar et al., 2022) and also provides an easy-to-use interface to add newly created PEFT methods to our framework. PEFT-Factory also includes all PEFT-Bench datasets, from which some are specifically adapted for autoregressive generation and classification (e.g., GLUE and SuperGLUE). In addition, PEFT-Factory implements the PSCP metric together with standard metrics and provides automated evaluation and results comparison.

Consequently, PEFT-Factory can be viewed as an instrument of PEFT-Bench that automates the training of different methods in a simple and configurable way. In this manner, PEFT-Factory takes just two configuration files (one for training and one for evaluation) as input, and returns fine-tuned adapter weights alongside the performance metrics. This specific property of PEFT-Factory allows an identical experimental environment during multiple runs (when equivalent hardware configurations are utilized). For this reason, we create a specific training and evaluation configuration for each model and method during the PEFT-Bench execution.

## 4 Experimental Setup

In all datasets, we utilize 10% randomly selected from the training set as a validation set and the original validation set as a test set for evaluation. During the validation phase, we measure validation loss and save the weights of the best validation loss for every 5% of the training steps. We train for 10 epochs with a batch size of 4. We utilize the AdamW (Loshchilov and Hutter, 2019) optimizer with a 0.1 warmup ratio, a weight decay of 1.0e-5, a learning rate of 5.0e-5, and a cosine learning rate scheduler. If not specified, we keep the training hyperparameters equal for all of the experiments and change only the PEFT method hyperparameters.

For $IA^3$, we apply adapters for key, value, and downsample feedforward modules of the transformer architecture. For Prompt Tuning, we fine-tune the soft prompt with a length of 100 tokens and initialize it randomly from the vocabulary of the trained model. For P-Tuning, we also fine-tune the soft prompt with a length of 100 tokens, but we reparametrize the soft prompts with an LSTM model with a hidden size of 768. For Prefix Tuning, we fine-tune the soft prompt with a length of 32 tokens, and we project the soft prompt with a small MLP with a hidden size of 512. For LoRA hyperparameters, we set the rank to 16, alpha to 16, and dropout to 0.05. Similarly to $IA^3$, we also reparametrize the key, value, and downsample matrices of the transformer architecture. For LNTuning, we leave all of the settings to their default values (the modules are selected based on the model architecture). For BitFit, we add the bias terms to the query and value matrices. For the stability experiments, we rerun training for 5 different random seeds, and we train each method for 20 epochs on each dataset.

As the *base model* for all PEFT methods, we adopt LLaMa3-8B-Instruct (containing exactly 8,030,261,248 trainable parameters) since this model is widely adopted, open-source, and supported by a strong community ecosystem. In addition, LLaMa3-8B-Instruct is a representative of modern instruction-tuned LLMs and provides a fair basis for benchmarking.

To calculate the PSCP metric, we calculate $P_t$ as an average of the performance metric (depending on the particular dataset) across all 5 runs. Since all performance metrics for each dataset are in the range $< 0, 1 >$, we can average the individual performance scores into a single value $P_t$ in a fair manner (as is already done in established benchmarks such as GLUE and SuperGLUE). We set $\beta$ for all factors to 1. For the efficiency factors, we set $M_p$ as the number of trainable parameters of the PEFT model (counting the parameters that are enabled in gradient descent). Consequently, $M_f$ is set as the difference between base model FLOPs and PEFT model FLOPs. Lastly, $M_m$ is set as an average maximum memory across all runs.

We set all reference constants $C$ to maximum values that would still be meaningful for training PEFT methods. More specifically, we set the number of trainable parameters $C_p = 5 \times 10^8$, as some of the smallest LLMs (e.g., Qwen2.5-0.5B (Yang et al., 2024)) contain 0.5 billion parameters, and if PEFT methods were to train such a number of parameters, it may be more beneficial to perform a full fine-tuning instead. We set the number of floating-point operations during inference $C_f = 10$ TFLOPs, approximately twice the number of the base model floating-point operations, as going beyond such a number will make inference with PEFT methods unacceptable. Finally, we set the training memory $C_m = 94$ GB, as it is typically the max-

| Method | # Trainable Parameters | GLUE (F1/Pearson) | | | | | | | | SuperGLUE (F1) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | MNLI | QQP | QNLI | SST-2 | STS-B | MRPC | RTE | CoLA | ReC | MRC | BQ | WiC | WSC | CB | COPA |
| | | llama-3-8b-instruct | | | | | | | | | | | | | | |
| Base | N/A | 59.2 | 0.5 | 80.4 | 92.2 | 67.4 | 78.8 | 69.7 | 78.3 | 49.4 | 68.0 | 80.8 | 66.6 | 43.6 | 47.0 | 73.8 |
| $IA^3$ | 196,608 | 91.1 | 86.3 | 94.4 | **96.4** | 88.3 | 86.7 | 84.5 | 88.9 | 86.0 | 87.0 | 89.7 | 72.1 | 42.0 | 68.9 | 96.6 |
| Prompt Tuning | 409,600 | 59.9 | 0.7 | 78.6 | 91.3 | 59.3 | 81.4 | 68.7 | 76.8 | 38.9 | 75.0 | 77.0 | 65.1 | 5.6 | 41.3 | 79.5 |
| Prefix Tuning | 34,177,536 | 48.0 | 12.7 | 92.1 | 91.6 | 82.8 | 38.8 | 72.5 | 45.5 | 26.9 | 45.5 | 72.2 | 60.1 | 0.0 | 49.8 | 7.5 |
| P-Tuning | 53,130,752 | 85.7 | 78.2 | 81.6 | 94.7 | 3.1 | 89.4 | 0.0 | 85.8 | 80.4 | 85.3 | 86.4 | 70.8 | 0.0 | 6.0 | 92.3 |
| LoRA | 14,680,064 | 91.1 | **88.2** | **96.1** | 95.9 | **90.7** | **91.0** | 85.7 | **89.7** | **90.3** | **88.8** | **91.0** | **75.2** | **53.6** | **83.9** | **97.8** |
| LNTuning | 266,240 | **91.2** | 87.1 | 95.1 | 96.1 | 89.8 | 86.8 | 84.4 | 89.3 | 88.2 | 87.3 | 89.7 | 74.2 | 48.4 | 67.6 | 97.7 |
| BitFit | 163,840 | 91.0 | 86.0 | 95.4 | **96.4** | 89.8 | 88.6 | **87.0** | 88.6 | 88.1 | 87.2 | 90.5 | 66.5 | 36.1 | 67.7 | 96.6 |

Table 1: Results of PEFT-Bench with different PEFT methods on NLU tasks from GLUE and SuperGLUE benchmarks. We measure Pearson correlation for STS-B and F1 for others. The best scores are in **bold**, and the second-best scores are underlined.

imum memory available on high-end GPU cards (e.g., Nvidia H100 NVL GPU). More details about the implementation and individual PEFT methods are provided in Appendix A.

## 5 PEFT-Bench Results

In PEFT-Bench, we first evaluate the PEFT methods on popular NLU datasets from GLUE and SuperGLUE, which can be found in Table 1. Consequently, we evaluate the PEFT methods on the other, math, and code generation tasks, with the results being shown in Table 2.

To properly account for different efficiency aspects of PEFT methods, we also evaluate them with the PSCP metric (introduced in Section 3.3). We gather costs for the number of parameters, inference FLOPs, and training memory, and apply the function from Equation 3. We report the PSCP results in Table 3 together with the cost values.

**LoRA achieves better performance, but BitFit and LNTuning are more efficient.** From the performance evaluation results in Tables 1 and 2, we can see that LoRA achieves the highest performance on many tasks. However, we can also see that a more parameter-efficient methods, BitFit and LNTuning, that train a smaller number of parameters and spend less GPU memory during training, were able to achieve better or on par performance in MNLI, SST-2, RTE and MMLU datasets. This efficiency difference was captured by our PSCP metric in Table 3, as it balanced the final result and offset the BitFit and LNTuning methods due to the lower number of parameters and lower memory consumption during training (compared to LoRA).

**PEFT learn the task structure but damages the correctness in math problem solving and code generation.** The results in Table 2 show a common pattern that PEFTs achieve better performance than the base model in classification or QA tasks. However, for problems that require more generation and reasoning, like GSM8K, Conala, CodeAlpacaPy, or APPS, we are witnessing that the PEFT actually harms the performance. This problem is, however, not affected by the math domain, since the performance on the MathQA and SVAMP datasets is generally increased after parameter-efficient fine-tuning. This may imply that other methods of alignment, like instruction tuning or reinforcement learning, are better suited for problems with longer sequences that require reasoning or chain-of-thought. Additionally, the trainable parameters may still be relatively low, for the model to enhance their reasoning capabilities during supervised fine-tuning.

**Soft prompt-based methods are harder to train.** The results in Tables 1 and 2 show that the soft prompt-based methods (i.e., Prompt Tuning, Prefix Tuning, and P-Tuning) perform generally worse compared to other methods, and in some cases, they achieve an F1 score of 0, mostly because they collapse into predicting only a single class. To achieve better results, more fine-grained per-dataset hyperparameter fine-tuning would probably be required. This suggests that out-of-the-box usability and versatility across different datasets are also important factors in designing an efficient method.

Additionally, we have observed memory spikes during training of the Prompt Tuning method. These spikes typically occurred during the first 1,000 steps of training, and they were significantly less noticeable when training on smaller amounts of data. After that, the memory usage was stable for the rest of the training. We believe this may be caused by the direct addition of a soft prompt into the input of an autoregressive model.

| Method | # Trainable Parameters | Others | | | | | | Math | | | Code Generation | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | MMLU | PIQA | SIQA | HS | WG | OBQA | MQA | GSM8K | SVAMP | Conala | CAPy | APPS |
| | | llama-3-8b-instruct | | | | | | | | | | | |
| Base | N/A | 57.1 | 46.0 | 70.3 | 62.2 | 9.6 | 78.3 | 26.8 | **79.2** | 56.7 | **31.2** | 32.7 | **21.6** |
| $IA^3$ | 196,608 | 64.6 | 86.0 | 79.2 | 91.7 | 81.4 | 83.7 | 44.3 | 68.3 | 85.3 | 23.9 | 32.0 | 18.6 |
| Prompt Tuning | 409,600 | 49.0 | 57.4 | 43.2 | 23.6 | 0.0 | 65.8 | 4.7 | 73.6 | 55.7 | 30.1 | 32.1 | 20.9 |
| Prefix Tuning | 34,177,536 | 15.4 | 45.3 | 32.0 | 24.9 | 27.5 | 50.1 | 15.5 | 37.8 | 85.7 | 27.5 | 9.1 | 15.2 |
| P-Tuning | 53,130,752 | 60.5 | 80.3 | 69.3 | 65.9 | 0.0 | 79.7 | 21.4 | 49.2 | 30.7 | 29.9 | 32.3 | 18.6 |
| LoRA | 14,680,064 | 63.1 | **88.9** | **81.5** | **94.8** | **88.2** | **87.7** | **49.4** | 69.1 | **88.0** | 28.9 | **35.0** | 20.7 |
| LNTuning | 266,240 | **65.8** | 86.4 | 81.0 | 92.1 | 81.9 | 85.3 | 44.6 | 68.3 | 85.7 | 26.4 | 32.6 | 18.6 |
| BitFit | 163,840 | 64.7 | 88.4 | 81.4 | 93.6 | 79.1 | 84.1 | 43.8 | 68.3 | 86.7 | 25.9 | 33.0 | 17.9 |

Table 2: Results of PEFT-Bench with different PEFT methods on QA, mathematical problem solving, and code generation tasks. We use accuracy for math problems, CodeBLEU (Ren et al., 2020) for code generation problems, and F1 for the others.

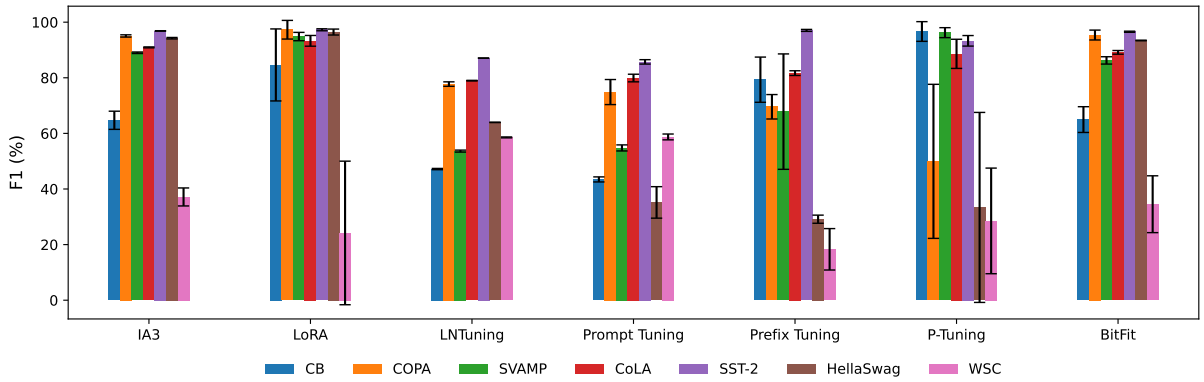


Figure 4: Bar chart showing the stability of different PEFT methods on 7 low-resource datasets. $IA^3$ achieves the lowest standard deviation on average across all datasets, and LoRA is less stable than other methods with CB datasets, while also achieving a better average score. Additionally, P-Tuning generally achieves the worst results in terms of performance and stability. For numerical results, see Table 11 in Appendix D.

| PEFT Method | $P_{avg}$ | $cost_p^{-1}$ | $cost_f^{-1}$ | $cost_m^{-1}$ | PSCP |
|---|---|---|---|---|---|
| $IA^3$ | 74.7 | 1.0 | 0.97 | 0.77 | 55.62 |
| Prompt Tuning | 50.0 | 1.0 | 0.86 | 0.69 | 29.49 |
| Prefix Tuning | 45.9 | 0.94 | 1.0 | 0.77 | 33.18 |
| P-Tuning | 51.7 | 0.9 | 0.86 | 0.73 | 29.26 |
| LoRA | 80.1 | 0.97 | 1.0 | 0.77 | 60.08 |
| LNTuning | 77.8 | 1.0 | 1.0 | 0.77 | **60.19** |
| BitFit | 75.3 | 1.0 | 1.0 | 0.80 | 60.10 |

Table 3: Results of the PEFT-Bench across different PEFT methods. LNTuning achieves a slightly better PSCP score than LoRA as it receives a lower penalty for the number of parameters.

**Stability experiments.** Finally, we also evaluate the stability of different PEFT methods. Since stability is not directly associated with the efficiency of PEFT methods, we decided to evaluate it separately from the previously-described performance and efficiency results. The results of the stability experiments can be seen in Figure 4. We evaluated the stability of individual PEFT methods on both, low (CB, COPA, SVAMP, CoLA, SST-2, HellaSwag, and WSC) as well as high resource (SST-2) datasets. Based on the results, the method that achieved the lowest standard deviations on average was $IA^3$. Moreover, we can see that most of the evaluated PEFT methods achieve relatively stable results on small data, except for LoRA on the CB and WSC dataset, where the standard deviations are larger compared to other methods.

A separate problem can be seen with P-Tuning, which is highly unstable within our experiment setup. This stability problem makes P-Tuning unusable with this setup and would probably require further hyperparameter tuning.

## 6 Conclusion

In our work, we introduce PEFT-Bench, the first unified and comprehensive benchmark for parameter-efficient fine-tuning methods for NLP, and PEFT-Factory, as an underlying technological framework supporting PEFT-Bench. PEFT-Bench

includes 27 datasets from various NLP problems and domains, implements support for off-the-shelf PEFT methods, simplifies the evaluation of new custom PEFT methods, and finally, introduces 7 PEFT methods as ready-to-use state-of-the-art baselines. We also propose the PSCP metric to properly incorporate efficiency factors like parameters, inference time, and memory usage into the evaluation.

In our comparison, we show that performance on downstream tasks should not be the only condition when evaluating PEFT methods, and sometimes, more efficient PEFT methods, like BitFit and LNTuning, can compensate for the performance with their efficiency. Additionally, we found that soft prompt-based methods are generally harder to train and require extensive hyperparameter tuning.

We view PEFT-Bench as a long-term, evolving initiative that will be continuously updated with new features and the addition of evaluations for different PEFT methods. The current state of PEFT-Bench, together with up-to-date results, is available at its GitHub repository[4]. In the future, we plan to use this benchmark to evaluate factors that affect the performance of multi-task PEFT methods.

## Acknowledgments

This work was partially funded by the European Union NextGenerationEU through the Recovery and Resilience Plan for Slovakia under the projects No. 09I01-03-V04-00006 and 09I01-03-V04-00068; and by the European Union, under the project LorAI - Low Resource Artificial Intelligence, GA No. 101136646.

Part of the research results was obtained using the computational resources procured in the national project funded by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90254); and the national project National competence centre for high performance computing (project code: 311070AKF2) funded by ERDF, EU Structural Funds Informatization of Society, Operational Program Integrated Infrastructure.

## Limitations

The evaluation of PEFT methods is primarily done on English datasets, which can potentially limit the PEFT-Bench to the English language. However, we believe that there is no specific benefit of evaluating PEFT methods for multilingual settings, as they are independent of the language of the fine-tuning data (unlike pre-trained LLMs, which were pre-trained on data of a specific language).

We also run PEFT-Bench on a single model – LLaMa3-8B-Instruct. This may limit the generalizability of PEFT-Bench, but, as with language, PEFT methods are not really dependent on the model architecture, and their performance would generally correlate with the model's performance.

Finally, we evaluate 7 different PEFT methods, with the majority being from the additive PEFT and soft prompt-based category. To keep experiments computationally feasible, we rely on the evaluated PEFT methods with limited hyperparameter tuning. Thanks to PEFT-Factory, the whole benchmark is easily extendable and can add their own PEFT method(s).

## Ethical Considerations

The experiments in this paper were conducted with publicly available datasets MNLI, QQP, QNLI, SST2, STS-B, MRPC, RTE, CoLA, MultiRC, ReCoRD, BoolQ, WiC, WSC, CB, COPA, MMLU, PIQA, SIQA, HellaSwag, WinoGrande, OpenBookQA, MathQA, GSM8K, SVAMP, Conala, CodeAlpacaPy, and APPS, citing the original authors. As we were unable to determine the licenses for all used datasets, we have opted to use them in the limited form possible, adhering to the terms of use of the GLUE and SuperGLUE benchmarks. As the datasets are commonly used in other related works and have been published in scientific works that underwent an established review process, we do not check for the presence of any offensive content, as it was already removed by the authors of these publicly available datasets. In addition, we do not utilize any personally identifiable information or offensive content, and we do not perform crowdsourcing in any form for data annotation. To our knowledge, we are not aware of any potential ethical harms or negative societal impacts of our work, apart from the ones related to the field of Machine Learning (i.e., the use of computational resources that are consuming energy and producing heat with indirect CO2 emission production). We follow the license terms for the LLaMa-3-8B-Instruct model we use – all models and datasets allow their use as part of the research. As we transform conditional generation into the classification problem (generating only labels), in most cases, we minimize the problem of generating offensive or biased content.

[4] https://github.com/kinit-sk/PEFT-Bench

**Impact Statement: CO2 Emissions Related to Experiments.** The experiments in this paper require a significant amount of GPU computing resources as we train and evaluate 1 model over multiple random initializations (5) for different methods (6) and datasets (27). Overall, the experiments, including evaluations (which did not require training, but still used GPU resources for inference) and preliminary experiments (which are not reported in the scope of our work), were conducted using a private infrastructure, which has a carbon efficiency of 0.432 kg$CO_2$eq/kWh. Approximately 3000 hours of computation were performed on hardware of type A100 PCIe 40GB (TDP of 250W). Total emissions are estimated to be 9.24 kg$CO_2$eq, of which 0 percent were directly offset. Whenever possible, we tried to reduce the computational costs.

## References


Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2357–2367, Minneapolis, Minnesota. Association for Computational Linguistics.

Akari Asai, Mohammadreza Salehi, Matthew Peters, and Hannaneh Hajishirzi. 2022. ATTEMPT: Parameter-efficient multi-task tuning via attentional mixtures of soft prompts. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6655–6672, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Roy Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second PASCAL recognising textual entailment challenge.

Robert Belanec, Simon Ostermann, Ivan Srba, and Maria Bielikova. 2025a. Task prompt vectors: Effective initialization through multi-task soft prompt transfer. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 77–94. Springer.

Robert Belanec, Ivan Srba, and Maria Bielikova. 2025b. Peft-factory: Unified parameter-efficient fine-tuning of autoregressive large language models. *arXiv preprint arXiv:2512.02764*.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland. Association for Computational Linguistics.

Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. 2009. The fifth PASCAL recognizing textual entailment challenge.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, and 1 others. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada. ACL.

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.

Charles W Cobb and Paul H Douglas. 1928. A theory of production. *The American economic review*, 18(1):139–165.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Proceedings of the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment*, MLCW'05, page 177–190, Berlin, Heidelberg. Springer-Verlag.

Marie-Catherine De Marneffe, Mandy Simons, and Judith Tonhauser. 2019. The commitmentbank: Investigating projection in naturally occurring discourse.

In *proceedings of Sinn und Bedeutung*, volume 23, pages 107–124.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, and 1 others. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235.

William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the International Workshop on Paraphrasing*.

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 preprint arXiv:2407.21783*.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Edward J Hu, Yelong Shen, 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*, 1(2):3.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, and 1 others. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. 2018. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 252–262.

Tushar Khot, Ashish Sabharwal, and Peter Clark. 2019. What's missing: A knowledge gap guided approach for multi-hop question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2814–2828, Hong Kong, China. Association for Computational Linguistics.

Pengxiang Lan, Haoyu Xu, Enneng Yang, Yuliang Liang, Guibing Guo, Jianzhe Zhao, and Xingwei Wang. 2025. Efficient and effective prompt tuning via prompt decomposition and compressed outer product. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4406–4421, Albuquerque, New Mexico. Association for Computational Linguistics.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Hector J Levesque, Ernest Davis, and Leora Morgenstern. 2011. The Winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, volume 46, page 47.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. 2023. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647*.

Vijay Lingam, Atula Tejaswi Neerkaje, Aditya Vavre, Aneesh Shetty, Gautham Krishna Gudur, Joydeep Ghosh, Eunsol Choi, Alex Dimakis, Aleksandar Bojchevski, and sujay sanghavi. 2024. SVFT: Parameter-efficient fine-tuning with singular vectors. In *2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ICML 2024)*.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022a. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023a. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, 55(9).

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022b. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–68, Dublin, Ireland. Association for Computational Linguistics.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2023b. Gpt understands, too. *AI Open*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

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`.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Mohammad Taher Pilehvar and Jose Camacho-Collados. 2019. WiC: the word-in-context dataset for evaluating context-sensitive meaning representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1267–1273, Minneapolis, Minnesota. Association for Computational Linguistics.

Nusrat Jahan Prottasha, Upama Roy Chowdhury, Shetu Mohanto, Tasfia Nuzhat, Abdullah As Sami, Md Shamol Ali, Md Shohanur Islam Sobuj, Hafijur Raman, Md Kowsher, and Ozlem Ozmen Garibay. 2025. Peft a2z: Parameter-efficient fine-tuning survey for large language and vision models. *arXiv preprint arXiv:2504.14117*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and 1 others. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of EMNLP*, pages 2383–2392. Association for Computational Linguistics.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. 2011. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *AAAI spring symposium: logical formalizations of commonsense reasoning*, pages 90–95.

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

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019. Social IQa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473, Hong Kong, China. Association for Computational Linguistics.

Zhengxiang Shi and Aldo Lipani. 2024. DePT: Decomposed prompt tuning for parameter-efficient fine-tuning. In *The Twelfth International Conference on Learning Representations*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on EMNLP*, pages 1631–1642.

Qi Sun, Edoardo Cetin, and Yujin Tang. 2025. Transformer-squared: Self-adaptive LLMs. In *The Thirteenth International Conference on Learning Representations*.

Pengwei Tang, Xiaolin Hu, and Yong Liu. 2025. ADePT: Adaptive decomposed prompt tuning for parameter-efficient fine-tuning. In *The Thirteenth International Conference on Learning Representations*.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, and 1 others. 2024. Gemini 1.5: Unlocking multimodal

understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Transactions of the ACL*, 7:625–641.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the ACL: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. ACL.

Yi Xin, Siqi Luo, Xuyang Liu, Haodi Zhou, Xinyu Cheng, Christina E Lee, Junlong Du, Haozhe Wang, MingCai Chen, Ting Liu, and 1 others. 2024. V-petl bench: A unified visual parameter-efficient transfer learning benchmark. *Advances in Neural Information Processing Systems*, 37:80522–80535.

Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. 2023. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *arXiv preprint arXiv:2312.12148*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 39 others. 2024. Qwen2 technical report. *ArXiv*, abs/2407.10671.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from stack overflow. In *Proceedings of the 15th international conference on mining software repositories*, pages 476–486.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy. Association for Computational Linguistics.

Jia-Chen Zhang, Yu-Jie Xiong, Chun-Ming Xia, Dong-Hai Zhu, and Xi-He Qiu. 2025a. Parameter-efficient fine-tuning of large language models via deconvolution in subspace. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 3924–3935, Abu Dhabi, UAE. Association for Computational Linguistics.

Pieyi Zhang, Richong Zhang, and Zhijie Nie. 2025b. Dynamic task vector grouping for efficient multi-task prompt tuning. *arXiv preprint arXiv:2503.18063*.

Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. 2018. Record: Bridging the gap between human and machine commonsense reading comprehension. *arXiv preprint arXiv:1810.12885*.

Yi-Kai Zhang, Lu Ren, Chao Yi, Qi-Wei Wang, De-Chuan Zhan, and Han-Jia Ye. 2023. Zhijian: A unifying and rapidly deployable toolbox for pre-trained model reuse. *arXiv preprint arXiv:2308.09158*.

Bingchen Zhao, Haoqin Tu, Chen Wei, Jieru Mei, and Cihang Xie. 2024. Tuning layernorm in attention: Towards efficient multi-modal LLM finetuning. In *The Twelfth International Conference on Learning Representations*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

# A Experiment Setup: Further Details

This appendix contains extended details about our experiment setup and training of PEFT methods for PEFT-Bench.

## A.1 Detailed Dataset Information

Table 4 shows a detailed comparison of information about datasets used in PEFT-Bench. In some cases, the dataset does not contain classes, which means that it either contains open questions (in case of question answering) or it is a generation task.

## A.2 Templates used for datasets

In this section, we provide instruction prompt templates that we have used while training for all of the datasets from GLUE (in Table 5), SuperGLUE (in Table 6), Others (in Table 7), and Math and Code Generation (in Table 8).

## B PSCP Importance Configurations

In this section, we provide an extended example of different configurations for the importance of each cost $\beta$. To evaluate the changes in the magnitude of the importance of each cost $\beta$, we provide results with different magnitudes in Table 9. The order of the methods can change if the performance differences between two methods are similar but differ only in a particular cost.

Based on the results from Table 10, we can see that with the change of importance for each cost, the order of the results can change (e.g., in Table 10a, the LoRA method achieves the best PSCP score, since we do not penalize the parameters cost as much as the other efficiency factors). Each subtable represents a different use case, where we emphasize different efficiency factors.

## C Descriptions of PEFT Methods

In this section, we provide a description of each PEFT method that we evaluate in this paper.

**IA**$^3$ (Infused Adapter by Inhibiting and Amplifying Inner Activation) (Liu et al., 2022a) introduces trainable scaling vectors that scale the activations of specific components within the transformer architecture. Instead of updating all model parameters, IA$^3$ inserts learnable vectors into the attention ($l_k$ and $l_v$) and feed-forward modules ($l_f$). The task is framed as learning the conditional probability $Pr(Y|X)$, parameterized by the frozen model weights $\theta$. IA$^3$ augments this with additional parameters $\theta_{IA^3}$, corresponding to the scaling vectors. During fine-tuning, only $\theta_{IA^3}$ is updated, while $\theta$ remains fixed, making the approach parameter-efficient.

**Prompt Tuning** (Lester et al., 2021) optimizes a soft prompt that is prepended to the input embeddings. The task is modeled as learning the conditional probability $Pr(Y|X)$, where $X$ denotes a sequence of input tokens and $Y$ represents the output tokens corresponding to the class label. In standard classification, this probability $Pr_\theta(Y|X)$ is parameterized by the model weights $\theta$. Prompting extends this by inserting an additional sequence of tokens (the prompt) $P$ before $X$, so the model instead maximizes $Pr_\theta(Y|[P;X])$, while keeping $\theta$ fixed. Prompt tuning further introduces parameters $\theta_P$ that define the prompt itself, and only $\theta_P$ is updated during training.

Simultaneous work **Prefix-Tunning** (Li and Liang, 2021) introduces similar trainable soft prompts that are prepended before key and value matrices of all attention layers. The soft prompts are not directly parametrized but rather generated by a multi-layer perceptron module.

**P-Tuning** (Liu et al., 2023b) considers inserting soft prompts before the inputs, using templates, which are essentially concatenated soft prompts with pre-trained embedding layers. P-Tuning also employs bi-directional LSTM to initialize the embeddings of the soft prompts.

Reparametrization is the process of transforming the original model parameters to other (reparametrized) parameters. In the context of PEFT, the transformation often lowers the number of trainable parameters. These reparametrized parameters can be transformed back to the original model parameter space, removing any overhead during inference. Building on the Intrinsic SAID citepaghajanyan-etal-2021-intrinsic method, **LoRA** (Hu et al., 2022) introduced two separate matrices $A \in \mathbb{R}^{r\times k}$ and $B \in \mathbb{R}^{d\times r}$ that form the resulting $\Delta W \in \mathbb{R}^{d\times k}$ matrix, given the initial (pre-trained) weight matrix $W_0 \in \mathbb{R}^{d\times k}$.

**LNTuning** (LayerNorm Tuning) (Zhao et al., 2024) adapts models by updating only the parameters of the layer normalization layers, while keeping all other weights frozen. As in standard classification, the task is modeled as learning the conditional probability $Pr(Y|X)$, parameterized by the model weights $\theta$. This enables efficient adaptation with minimal trainable parameters. Additionally, LNTuning does not create any overhead during inference, since it does not add any additional parameters to the model.

**BitFit** (Bias Fine-tuning) (Ben Zaken et al., 2022) fine-tunes only the bias terms of weights in selected modules of transformer architecture. However, in modern LLMs, the bias term is often skipped while designing the model. In our implementation, instead of reusing the pre-trained bias term (as in the original implementation), we replace the linear layer with a new one that also contains the bias term and copy the pre-trained weights.

## D Additional results

In this section, we provide additional results to complement the main results in the body of the paper. Table 11 shows the numerical results from Figure 4 in Section 5.

| Area | Dataset | Task | #Classes | #Train | #Test |
|---|---|---|---|---|---|
| | *GLUE Benchmark* | | | | |
| NLU and Reasoning | MNLI | Natural language inference | 3 | 392,702 | 9,815 |
| | QQP | Paraphrase classificaiton | 2 | 363,846 | 40,430 |
| | QNLI | Natural language inference | 2 | 104,743 | 5,463 |
| | SST-2 | Sentiment classification | 2 | 67,349 | 872 |
| | STS-B | Sentence similarity | N/A | 5,749 | 1,500 |
| | MRPC | Paraphrase classificaiton | 2 | 3,668 | 408 |
| | RTE | Natural language inference | 2 | 2,490 | 277 |
| | CoLA | Acceptability classification | 2 | 8,551 | 1,043 |
| | *SuperGLUE Benchmark* | | | | |
| | MultiRC | Question answering | 2 | 27,243 | 4,848 |
| | ReCoRD | Question answering | N/A | 100,730 | 10,000 |
| | BoolQ | Question answering | 2 | 9,427 | 3,270 |
| | WiC | Word sense disambiguation | 2 | 5,428 | 638 |
| | WSC | Coreference resolution | 2 | 554 | 104 |
| | CB | Natural language inference | 3 | 250 | 56 |
| | COPA | Question answering | 2 | 400 | 100 |
| | *Others* | | | | |
| | MMLU | Question answering | 4 | 99,842 | 1,531 |
| | PIQA | Question answering | 2 | 16,113 | 1,838 |
| | SIQA | Question answering | 3 | 33,410 | 1,954 |
| | HellaSwag | Natural language inference | 4 | 39,905 | 10,042 |
| | WinoGrande | Commonsense reasoning | 2 | 40,398 | 1,267 |
| | OBQA | Question answering | 4 | 4,957 | 500 |
| Math | MathQA | Question answering | 5 | 29,837 | 4,475 |
| | GSM8K | Math word problems | N/A | 7,473 | 1,319 |
| | SVAMP | Simple math problems | N/A | 700 | 300 |
| Code generation | Conala | Text-to-code | N/A | 2,379 | 500 |
| | CodeAlpacaPy | Text-to-code | N/A | 8,477 | 942 |
| | APPS | Text-to-code | N/A | 5,000 | 5,000 |

Table 4: Comparison of datasets used in PEFT-Bench. Each dataset has its own task assigned, totaling 12 unique tasks. The comparison contains the number of classes (if applicable), train sizes, and test sizes for each dataset.

| Dataset | Template |
|---|---|
| MNLI | Classify the following premise and hypothesis pair into entailment, neutral, and contradiction classes.<br>Respond only with the corresponding class.<br><br>{premise}<br><br>{hypothesis}<br><br>{label} |
| QQP | Classify the following question pair into duplicate and not_duplicate classes.<br>Respond only with the corresponding class.<br><br>{question1}<br><br>{question2}<br><br>{label} |
| QNLI | Classify the following question and sentence pair into entailment and not_entailment classes.<br>Respond only with the corresponding class.<br><br>{question}<br><br>{sentence}<br><br>{label} |
| SST-2 | Classify the following sentence into negative and positive classes.<br>Respond only with the corresponding class.<br><br>{sentence}<br><br>{label} |
| STS-B | Assign a similarity score from 0 to 5 to the following sentence pair.<br>Respond only with the corresponding score.<br><br>{sentence1}<br><br>{sentence2}<br><br>{label} |
| MRPC | Classify the following sentence pair into not_equivalent and equivalent classes.<br>Respond only with the corresponding class.<br><br>{sentence1}<br><br>{sentence2}<br><br>{label} |
| RTE | Classify the following sentence pair into entailment and not_entailment classes.<br>Respond only with the corresponding class.<br><br>{sentence1}<br><br>{sentence2}<br><br>{label} |
| CoLA | Classify the following sentence into unacceptable and acceptable classes.<br>Respond only with the corresponding class.<br><br>{sentence}<br><br>{label} |

Table 5: Instruction prompt templates for each dataset in the GLUE benchmark.

| Dataset | Template |
|---|---|
| MultiRC | Based on the following paragraph, question, and answer, determine whether the answer answers the question.<br>Respond only with the corresponding true or false.<br><br>Paragraph: {paragraph}<br><br>Question: {question}<br><br>Answer: {answer}<br><br>{label} |
| ReCoRD | Based on the following query, entities, and passage, replace the @placeholder in the query.<br>Respond only with the correct replacement.<br><br>Query: {query}<br><br>Entities: {hypothesis}<br><br>Passage: {passage}<br><br>{label} |
| BoolQ | Based on the passage, respond to the following question with true or false.<br>Respond only with the corresponding true or false.<br><br>{question}<br><br>{passage}<br><br>{label} |
| WiC | Based on the context from the following sentence pair, classify the word into true and false classes,<br>based on its meaning in both of the sentences. Respond only with the corresponding true or false.<br><br>Sentence1: {sentence1}<br><br>Sentence2: {sentence2}<br><br>Word: {word}<br><br>{label} |
| WSC | Based on the following sentence, determine whether the pronoun marked with * * is referencing<br>the noun marked with # #. Respond only with the corresponding true and false.<br><br>{sentence}<br><br>{label} |
| CB | Classify the following premise and hypothesis pair into entailment, contradiction, and neutral classes.<br>Respond only with the corresponding class.<br><br>{premise}<br><br>{hypothesis}<br><br>{label} |
| COPA | Based on the following premise and choice pair, select the plausible alternative from the choice pair.<br>Respond only with the corresponding choice1 or choice2.<br><br>Premise: {premise}<br><br>Choice1: {choice1}<br><br>Choice2: {choice2}<br><br>{label} |

Table 6: Instruction prompt templates for each dataset in the SuperGLUE benchmark.

| Dataset | Template |
|---|---|
| MMLU | Based on the question and provided choices, select the right answer.<br>Respond only with the corresponding choices A, B, C, and D.<br><br>Question: {question}<br><br>Choices:<br>A:{choice A}<br>B:{choice B}<br>C:{choice C}<br>D:{choice D}<br><br>{label} |
| PIQA | Based on the question and provided solutions, select the right solution.<br>Respond only with the corresponding solution1 or solution2.<br><br>Question: {question}<br><br>Solution1: {solution1}<br><br>Solution2: {solution2}<br><br>{label} |
| SIQA | Based on the context, question, and provided choices, select the right answer.<br>Respond only with the corresponding choices A, B, and C.<br><br>Context: {context}<br><br>Question: {question}<br><br>Choices:<br>A:{choice A}<br>B:{choice B}<br>C:{choice C}<br><br>{label} |
| HellaSwag | Based on the sentence and provided endings, select the correct ending that finishes the sentence correctly.<br>Respond only with the corresponding ending1, ending2, ending3, and ending4.<br><br>Sentence: {sentence}<br><br>Ending1: {ending1}<br>Ending2: {ending2}<br>Ending3: {ending3}<br>Ending4: {ending4}<br><br>{label} |
| WinoGrande | Based on the sentence and provided options, select the best option that replaces the '_' character<br>in the sentence. Respond only with the corresponding option1 or option2.<br><br>Sentence: {sentence}<br><br>Option1: {option1}<br><br>Option2: {option2}<br><br>{label} |
| OBQA | Based on the question and provided choices, select the right answer.<br>Respond only with the corresponding choices A, B, C, and D.<br><br>Question: {question}<br><br>Choices:<br>A:{choice A}<br>B:{choice B}<br>C:{choice C}<br>D:{choice D}<br><br>{label} |

Table 7: Instruction prompt templates for each dataset in the Others category of datasets.

| Dataset | Template |
|---|---|
| MathQA | Based on the problem and provided options, select the right answer.<br>Respond only with the corresponding choices a, b, c, d, and e.<br><br>Problem: {problem}<br><br>Options: a ) {option A} , b ) {option B} , c ) {option C} , d ) {choice D} , e ) {option E}<br><br>{label} |
| GSM8K | Answer following math question. The answer is always numerical.<br>Mark your final answer with a newline starting with '####'.<br><br>{question}<br><br>{answer} |
| SVAMP | Answer following math question. The answer is always numerical.<br>Respond only with a single equation with parentheses (e.g., (1 + 1) = 2).<br><br>Question: {question}<br><br>{answer} |
| Conala | Based on the following instruction, generate a valid Python code that answers it.<br><br>{rewritten_intent}<br><br>{snippet} |
| CodeAlpacaPy | Based on the following instruction, generate a valid Python code that answers it.<br><br>{prompt}<br><br>{response} |
| APPS | Based on the following instruction, generate a valid Python code that answers it.<br><br>{prompt}<br><br>{response} |

Table 8: Instruction prompt templates for each dataset in the Math and Code Generation category of datasets.

| PEFT Method | $P_{avg}$ | $cost_p^{-1}$ | $cost_f^{-1}$ | $cost_m^{-1}$ | PSCP |
|---|---|---|---|---|---|
| $IA^3$ | 74.7 | 1.0 | 0.97 | 0.77 | 55.62 |
| Prompt Tuning | 50.0 | 1.0 | 0.86 | 0.69 | 29.49 |
| Prefix Tuning | 45.9 | 0.94 | 1.0 | 0.77 | 33.18 |
| P-Tuning | 51.7 | 0.9 | 0.86 | 0.73 | 29.26 |
| LoRA | 80.1 | 0.97 | 1.0 | 0.77 | 60.08 |
| LNTuning | 77.8 | 1.0 | 1.0 | 0.77 | **60.19** |
| BitFit | 75.3 | 1.0 | 1.0 | 0.8 | <u>60.1</u> |

(a)

| PEFT Method | $P_{avg}$ | $cost_p^{-2}$ | $cost_f^{-2}$ | $cost_m^{-2}$ | PSCP |
|---|---|---|---|---|---|
| $IA^3$ | 74.7 | 1.0 | 0.94 | 0.59 | 41.41 |
| Prompt Tuning | 50.0 | 1.0 | 0.73 | 0.47 | 17.39 |
| Prefix Tuning | 45.9 | 0.88 | 1.0 | 0.6 | 23.98 |
| P-Tuning | 51.7 | 0.82 | 0.73 | 0.53 | 16.56 |
| LoRA | 80.1 | 0.94 | 1.0 | 0.6 | 45.06 |
| LNTuning | 77.8 | 1.0 | 1.0 | 0.6 | <u>46.57</u> |
| BitFit | 75.3 | 1.0 | 1.0 | 0.64 | **47.97** |

(b)

| PEFT Method | $P_{avg}$ | $cost_p^{-3}$ | $cost_f^{-3}$ | $cost_m^{-3}$ | PSCP |
|---|---|---|---|---|---|
| $IA^3$ | 74.7 | 1.0 | 0.91 | 0.45 | 30.83 |
| Prompt Tuning | 50.0 | 1.0 | 0.63 | 0.33 | 10.26 |
| Prefix Tuning | 45.9 | 0.82 | 1.0 | 0.46 | 17.33 |
| P-Tuning | 51.7 | 0.74 | 0.63 | 0.39 | 9.37 |
| LoRA | 80.1 | 0.92 | 1.0 | 0.46 | 33.8 |
| LNTuning | 77.8 | 1.0 | 1.0 | 0.46 | <u>36.03</u> |
| BitFit | 75.3 | 1.0 | 1.0 | 0.51 | **38.29** |

(c)

| PEFT Method | $P_{avg}$ | $cost_p^{-4}$ | $cost_f^{-4}$ | $cost_m^{-4}$ | PSCP |
|---|---|---|---|---|---|
| $IA^3$ | 74.7 | 1.0 | 0.89 | 0.35 | 22.95 |
| Prompt Tuning | 50.0 | 1.0 | 0.54 | 0.23 | 6.05 |
| Prefix Tuning | 45.9 | 0.77 | 1.0 | 0.36 | 12.53 |
| P-Tuning | 51.7 | 0.67 | 0.54 | 0.29 | 5.31 |
| LoRA | 80.1 | 0.89 | 1.0 | 0.36 | 25.35 |
| LNTuning | 77.8 | 1.0 | 1.0 | 0.36 | <u>27.88</u> |
| BitFit | 75.3 | 1.0 | 1.0 | 0.41 | **30.56** |

(d)

Table 9: Results with PSCP metric under equal importance of each cost with different magnitudes. Because the performance differences between BitFit and LNTuning are small, the relative importance of each cost changed the ordering of the methods.

| PEFT Method | $P_{avg}$ | $cost_p^{-0.5}$ | $cost_f^{-1}$ | $cost_m^{-1}$ | PSCP |
|---|---|---|---|---|---|
| $IA^3$ | 74.7 | 1.0 | 0.97 | 0.77 | 55.63 |
| Prompt Tuning | 50.0 | 1.0 | 0.86 | 0.69 | 29.5 |
| Prefix Tuning | 45.9 | 0.97 | 1.0 | 0.77 | 34.29 |
| P-Tuning | 51.7 | 0.95 | 0.86 | 0.73 | 30.78 |
| LoRA | 80.1 | 0.99 | 1.0 | 0.77 | **60.95** |
| LNTuning | 77.8 | 1.0 | 1.0 | 0.77 | 60.21 |
| BitFit | 75.3 | 1.0 | 1.0 | 0.8 | 60.11 |

(a)

| PEFT Method | $P_{avg}$ | $cost_p^{-1}$ | $cost_f^{-0.5}$ | $cost_m^{-1}$ | PSCP |
|---|---|---|---|---|---|
| $IA^3$ | 74.7 | 1.0 | 0.98 | 0.77 | 56.47 |
| Prompt Tuning | 50.0 | 1.0 | 0.93 | 0.69 | 31.85 |
| Prefix Tuning | 45.9 | 0.94 | 1.0 | 0.77 | 33.19 |
| P-Tuning | 51.7 | 0.9 | 0.93 | 0.73 | 31.63 |
| LoRA | 80.1 | 0.97 | 1.0 | 0.77 | 60.08 |
| LNTuning | 77.8 | 1.0 | 1.0 | 0.77 | **60.19** |
| BitFit | 75.3 | 1.0 | 1.0 | 0.8 | 60.1 |

(b)

| PEFT Method | $P_{avg}$ | $cost_p^{-1}$ | $cost_f^{-1}$ | $cost_m^{-0.5}$ | PSCP |
|---|---|---|---|---|---|
| $IA^3$ | 74.7 | 1.0 | 0.97 | 0.88 | 63.47 |
| Prompt Tuning | 50.0 | 1.0 | 0.86 | 0.83 | 35.53 |
| Prefix Tuning | 45.9 | 0.94 | 1.0 | 0.88 | 37.73 |
| P-Tuning | 51.7 | 0.9 | 0.86 | 0.86 | 34.22 |
| LoRA | 80.1 | 0.97 | 1.0 | 0.88 | 68.37 |
| LNTuning | 77.8 | 1.0 | 1.0 | 0.88 | **68.41** |
| BitFit | 75.3 | 1.0 | 1.0 | 0.89 | 67.26 |

(c)

| PEFT Method | $P_{avg}$ | $cost_p^{-2}$ | $cost_f^{-1}$ | $cost_m^{-1}$ | PSCP |
|---|---|---|---|---|---|
| $IA^3$ | 74.7 | 1.0 | 0.97 | 0.77 | 55.59 |
| Prompt Tuning | 50.0 | 1.0 | 0.86 | 0.69 | 29.46 |
| Prefix Tuning | 45.9 | 0.88 | 1.0 | 0.77 | 31.05 |
| P-Tuning | 51.7 | 0.82 | 0.86 | 0.73 | 26.45 |
| LoRA | 80.1 | 0.94 | 1.0 | 0.77 | 58.37 |
| LNTuning | 77.8 | 1.0 | 1.0 | 0.77 | **60.16** |
| BitFit | 75.3 | 1.0 | 1.0 | 0.8 | 60.08 |

(d)

| PEFT Method | $P_{avg}$ | $cost_p^{-1}$ | $cost_f^{-2}$ | $cost_m^{-1}$ | PSCP |
|---|---|---|---|---|---|
| $IA^3$ | 74.7 | 1.0 | 0.94 | 0.77 | 53.94 |
| Prompt Tuning | 50.0 | 1.0 | 0.73 | 0.69 | 25.27 |
| Prefix Tuning | 45.9 | 0.94 | 1.0 | 0.77 | 33.14 |
| P-Tuning | 51.7 | 0.9 | 0.73 | 0.73 | 25.05 |
| LoRA | 80.1 | 0.97 | 1.0 | 0.77 | 60.08 |
| LNTuning | 77.8 | 1.0 | 1.0 | 0.77 | **60.19** |
| BitFit | 75.3 | 1.0 | 1.0 | 0.8 | 60.1 |

(e)

| PEFT Method | $P_{avg}$ | $cost_p^{-1}$ | $cost_f^{-1}$ | $cost_m^{-2}$ | PSCP |
|---|---|---|---|---|---|
| $IA^3$ | 74.7 | 1.0 | 0.97 | 0.59 | 42.71 |
| Prompt Tuning | 50.0 | 1.0 | 0.86 | 0.47 | 20.31 |
| Prefix Tuning | 45.9 | 0.94 | 1.0 | 0.6 | 25.64 |
| P-Tuning | 51.7 | 0.9 | 0.86 | 0.53 | 21.4 |
| LoRA | 80.1 | 0.97 | 1.0 | 0.6 | 46.39 |
| LNTuning | 77.8 | 1.0 | 1.0 | 0.6 | 46.6 |
| BitFit | 75.3 | 1.0 | 1.0 | 0.64 | **47.99** |

(f)

Table 10: Results with PSCP metric with different importance of each cost $\beta$. Notice how the order of the methods changes in Table (a) when we decrease the importance of the cost for the number of trainable parameters $\beta_p$.

| | CB | COPA | SVAMP | CoLA | SST-2 | HellaSwag | WSC |
|---|---|---|---|---|---|---|---|
| $IA^3$ | $64.7 \pm 3.3$ | $95.0 \pm 0.4$ | $89.0 \pm 0.3$ | $90.9 \pm 0.2$ | $96.8 \pm 0.1$ | $94.2 \pm 0.3$ | $37.1 \pm 3.2$ |
| LoRA | $84.6 \pm 13.0$ | $97.3 \pm 3.4$ | $94.8 \pm 1.5$ | $93.3 \pm 1.9$ | $97.3 \pm 0.4$ | $96.4 \pm 1.1$ | $24.2 \pm 25.8$ |
| LNTuning | $47.2 \pm 0.2$ | $77.7 \pm 0.8$ | $53.6 \pm 0.4$ | $79.0 \pm 0.1$ | $87.1 \pm 0.0$ | $64.0 \pm 0.1$ | $58.6 \pm 0.2$ |
| Prompt Tuning | $43.4 \pm 0.9$ | $74.9 \pm 4.5$ | $54.8 \pm 1.1$ | $79.9 \pm 1.4$ | $85.7 \pm 0.8$ | $35.2 \pm 5.7$ | $58.7 \pm 1.0$ |
| Prefix Tuning | $79.3 \pm 8.1$ | $69.6 \pm 4.4$ | $67.8 \pm 20.8$ | $81.7 \pm 0.9$ | $97.0 \pm 0.4$ | $29.1 \pm 1.5$ | $18.3 \pm 7.5$ |
| P-Tuning | $96.6 \pm 3.5$ | $49.9 \pm 27.7$ | $96.2 \pm 1.8$ | $88.6 \pm 5.2$ | $93.3 \pm 1.9$ | $33.4 \pm 34.2$ | $28.5 \pm 19.0$ |
| BitFit | $65.0 \pm 4.6$ | $95.4 \pm 1.8$ | $86.2 \pm 1.3$ | $89.1 \pm 0.7$ | $96.6 \pm 0.2$ | $93.4 \pm 0.1$ | $34.5 \pm 10.2$ |

Table 11: Results from stability experiments. These are numerical results from the graph in Figure 4.