PEFT-FACTORY: Unified Parameter-Efficient Fine-Tuning of Autoregressive Large Language Models

Robert Belanec[♠,†], **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`
Demonstration video: **`https://youtu.be/Q3kxvlyO-XY`**
Installation package: **`https://pypi.org/project/peftfactory`**
Live demo: **`https://peftfactory.kinit.sk`**

Abstract

Parameter-Efficient Fine-Tuning (PEFT) methods address the increasing size of Large Language Models (LLMs). Currently, many newly introduced PEFT methods are challenging to replicate, deploy, or compare with one another. To address this, we introduce PEFT-FACTORY, a unified framework for efficient fine-tuning LLMs using both off-the-shelf and custom PEFT methods. While its modular design supports extensibility, it natively provides a representative set of 19 PEFT methods, 27 classification and text generation datasets addressing 12 tasks, and both standard and PEFT-specific evaluation metrics. As a result, PEFT-FACTORY provides a ready-to-use, controlled, and stable environment, improving replicability and benchmarking of PEFT methods. PEFT-FACTORY is a downstream framework that originates from the popular LLaMA-Factory, and is publicly available at `https://github.com/kinit-sk/PEFT-Factory`.

1 Introduction

Large Language Models (LLMs) (Minaee et al., 2024; Radford et al., 2019; Dubey et al., 2024; Raffel et al., 2020) achieved remarkable results in many different Natural Language Processing (NLP) tasks mainly after the introduction of the transformer architecture (Vaswani, 2017). However, for its great scaling capabilities (Kaplan et al., 2020), the size of the model in terms of trainable parameters is continuously increasing accordingly. The growing number of LLM parameters rendered fine-tuning computationally expensive, data-hungry, and hardly accessible for many researchers and practitioners.

Parameter-Efficient Fine-Tuning (PEFT) methods (Xu et al., 2023; Ding et al., 2023; Lialin et al., 2023; Han et al., 2024) aim to address these issues by training only a small percentage of the full model's parameters while achieving performance comparable to that of full model fine-tuning. Such a decrease in trainable parameters can be achieved by adding new parameters (Houlsby et al., 2019), selecting specific parameters (Ben Zaken et al., 2022) for training, or by reparameterizing the model with a smaller number of parameters (Hu et al., 2022).

Due to their effectiveness, PEFT methods have gained popularity and become an attractive research area, with many new contributions being introduced each year (Han et al., 2024). However, the large number of newly introduced PEFT methods makes it harder to compare their contributions, resulting in only a few established PEFT methods (LoRA variants in most cases) being used in practice, while others, which may be more effective, remain largely unused. Moreover, many new PEFT methods lack a fully functional open-source implementation, and essential details on the experimental setup often prevent fellow researchers from replicating their results. Therefore, many researchers (Asai et al., 2022; Shi and Lipani, 2024; Tang et al., 2025) have to rely on reported performance metrics, while there is a risk of not replicating exactly the same experimental setup for their own methods (making a comparison potentially unfair), as well as it is not feasible to rerun the existing solutions on additional datasets/tasks. Lastly, many established PEFT methods lack proper evaluation on autoregressive LLMs.

To tackle these accumulating problems, **we introduce PEFT-FACTORY, an easy-to-use and modular framework for efficient fine-tuning and evaluation of LLMs using different PEFT methods**. PEFT-FACTORY is based on the popular and open-source fine-tuning framework *LLaMA-Factory* (Zheng et al., 2024b). It is built using PyTorch (Paszke et al., 2019) and utilizes open-source Python modules for training LLMs, including Transformers (Wolf et al., 2020), PEFT (Mangrulkar et al., 2022), TRL (von Werra et al., 2020), and Adapters (Poth et al., 2023).

Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics
Volume 3: System Demonstrations, pages 188–202

Our main contributions are as follows:

- PEFT-Factory provides a support for off-the-shelf methods from popular PEFT provider frameworks like *HuggingFace PEFT* (Mangrulkar et al., 2022) or *Adapters* (Poth et al., 2023) as well as dynamic loading of *custom user-created PEFT methods*. In contrast to the existing solutions, it provides so-far-missing support for *soft prompt-based*, *adapter-based* and *selective* PEFT methods; as well as for *classification* tasks.

- PEFT-Factory natively provides a representative set of *19 PEFT methods*, *27 classification and text generation datasets* addressing *12 unique tasks*, and standard as well as PEFT-specific *evaluation metrics*. This ready-to-use setup enables quick adoption and experimentation by researchers and practitioners, significantly improving the currently limited replicability and benchmarking of PEFT methods.

- PEFT-Factory is designed with future *extensibility* in mind and provides a fully open-source codebase for anyone to use. It implements a standardized PEFT interface to enable modular addition of newly created PEFT methods. Similarly, it allows easy extension for additional datasets.

2 Related Work

There has been a significant rise in the number of frameworks used for training of (not only) LLMs. LLaMA-Factory (Zheng et al., 2024b) is a recent addition to such frameworks and offers end-to-end and easy-to-use training of LLMs ranging across all of the stages (from pre-training to alignment via reinforcement learning). LLaMA-Factory also provides a graphical user interface called LLaMABoard, implemented in Gradio (Abid et al., 2019), which enhances the ease of use of LLaMA-Factory. Despite being a really popular and useful tool for LLM training, LLaMA-Factory still provides fine-tuning only with LoRA (Hu et al., 2022) and its variants, namely QLoRA (Dettmers et al., 2023), DoRA (Liu et al., 2024), LoRA+ (Hayou et al., 2024), PiSSA (Meng et al., 2024), and GaLore (Zhao et al., 2024b). With the recent update, LLaMA-Factory also allows Orthogonal Fine-Tuning (OFT) (Qiu et al., 2023), which utilizes the Cayley transformation (Cayley, 1846) to fine-tune only orthogonal vectors. Nevertheless, the selection of PEFT methods in LLaMA-Factory still remains limited. Lastly, LLaMA-Factory primarily focuses on text-generation problems and does not incorporate the possibility of casting text-generation problems as classification tasks. Our framework PEFT-Factory addresses both the limited number of available PEFT methods and the potential for fine-tuning LLMs for classification.

There are also other LLM training frameworks that are less easy to run (compared to LLaMA-Factory) and have their specific benefits. FastChat (Zheng et al., 2023) is a specialized framework for training LLMs for chat-completion. LitGPT (AI, 2023) and LMFlow (Diao et al., 2023) are extensible and convenient general training frameworks that support various generative models and training methods. Axolotl (Axolotl maintainers and contributors, 2023) is a terminal-based tool for efficient post-training of LLMs without sacrificing functionality or scale. Open-Instruct (Wang et al., 2023a) focuses on instruction fine-tuning for LLMs and provides multiple models and recipes for this purpose. H2O LLM Studio[1] is a more enterprise-oriented, all-in-one tool that also provides a graphical interface for developing and deploying LLM models. GPT4All (Anand et al., 2023) creates a user-friendly interface around llamacpp. ColossalAI (Li et al., 2023) focuses on delivering a framework for distributed fine-tuning.

In addition, LLaMA-Adapter (Zhang et al., 2023) and LLaMA-Accesory (Gao et al., 2023) are more lightweight frameworks, where LLaMA-Adapter adds trainable adapters to (not only) LLaMA models and LLaMA-Accesory provides a full toolkit for LLM development. LLaMA-Adapter is often implemented in previously-mentioned frameworks, such as LitGPT.Table 1 provides a summary of unique PEFT-Factory features when compared with popular fine-tuning frameworks as well as our upstream framework LLaMA-Factory. Based on our analysis of related frameworks and to the best of our knowledge, we have identified 3 key features that are currently missing or limited, and are novel in our work: 1) training of LLMs with other than reprametrization-based PEFT methods, 2) modular and dynamic addition of new PEFT methods, and 3) support for training and evaluation of LLMs for classification.

[1] `https://github.com/h2oai`

	Reparametrized	Soft Prompt-Based	Adapter-Based	Selective	Classification Datasets	Classification Metrics	Extensibility
PEFT-FACTORY	8	5	4	2	✔	✔	datasets, models, PEFT methods
LLaMA-Factory	7	0	0	0	✘	✘	datasets, models
FastChat	3	0	0	0	✘	✘	datasets, models
LitGPT	2	0	0	0	✔	✔	datasets, models
LMFlow	3	0	0	0	✘	✘	datasets, models
Axolotl	2	0	0	0	✘	✘	datasets, models
Open-Instruct	3	0	0	0	✘	✘	✘
H2O LLM Studio	3	0	0	0	✔	✔	datasets, models
GPT4All	2	0	0	0	✘	✘	✘

Table 1: Comparison of PEFT-FACTORY to popular fine-tuning frameworks. Only PEFT-FACTORY allows for out-of-the-box non-reparametrization efficient fine-tuning with the extensibility of additional and custom fine-tuning methods. Comparison at the level of individual PEFT methods can be found in Table 3 of Appendix B.

3 PEFT-FACTORY

The PEFT-FACTORY consists of four main components: 1) PEFT Methods, 2) Datasets, 3) Models, and 4) Metrics, as also depicted in Figure 1.

In the *PEFT methods* component, we design and implement support for reparameterized, soft prompt-based, adapter-based, and selective PEFT methods, from HuggingFace PEFT (Mangrulkar et al., 2022) and Adapters (Poth et al., 2023) PEFT provider frameworks. We also provide a custom PEFT interface for more advanced users to provide and dynamically load their custom PEFT methods into PEFT-FACTORY. Currently, we include *19 different PEFT methods* (out of them, 7 are natively provided by the LLaMA-Factory). Full listing of PEFT methods covered by PEFT-FACTORY can be found in Table 3 of Appendix B.

The core of the *Datasets* component is the dataset loader supporting datasets from classification tasks, with the possibility of adding separate instructions for instruction fine-tuned models (a missing feature of LLaMA-Factory). Additionally, we include and adapt multiple well-known classification benchmarks, as well as text-generation tasks, totalling *27 datasets*.

Regarding *Models*, PEFT-FACTORY leverages the existing support provided by LLaMA-Factory. It enables the utilization of a wide range of models from different model families, spanning from 0.5 (e.g., Qwen 2.5 (Yang et al., 2024)) to 671 (e.g., DeepSeek R1 (Guo et al., 2025)) billion parameters. For demonstration purposes, we selected Llama-3.2-1B-Instruct (Dubey et al., 2024) as it is a popular representative of a reasonable size that allows fast training to demonstrate PEFT-FACTORY.

Within the *Metrics* component, we add classification and performance-based metrics into the evaluation of LLMs trained using PEFT methods. This includes the addition of standard classification metrics, such as accuracy and F1, as well as the PSCP metric (Belanec et al., 2025), which incorporates various efficiency factors into the results.

3.1 Off-The-Shelf PEFT Support

There are many different PEFT methods included off-the-shelf within the PEFT provider libraries, such as HuggingFace PEFT and Adapters. In the current state, we include 10 different off-the-shelf PEFT methods that we have tested with different state-of-the-art LLMs, namely, from the **Adapters library** – *Parallel Adapter* (He et al., 2022), *Bottleneck Adapter* (Houlsby et al., 2019), and *Sequential Bottleneck Adapter* (Pfeiffer et al., 2020); and from the **HuggingFace PEFT** – *Prompt Tuning* (Lester et al., 2021), *Prefix Tuning* (Li and Liang, 2021), *P-Tuning* (Liu et al., 2023), *P-Tuningv2* (Liu et al., 2022b), *MTP* (Wang et al., 2023b), *LNTuning* (Zhao et al., 2024a), and *IA*^{3} (Liu et al., 2022a). Moreover, PEFT-FACTORY enables to easily *add more of such off-the-shelf PEFT methods simply by updating two constants in the code*.

From the implementation perspective, to include support for these libraries in PEFT-FACTORY, we created a unified `PeftArguments` class that inherits both the `PEFTConfig` and `AdapterConfig` classes for typing purposes. This joint class is then used for parsing the parameters from configurations via `HFArgumentParser`. We store all supported off-the-shelf PEFT methods in several constants, specifically lists `HF_PEFT_METHODS` and `ADAPTERS_METHODS`, along with their counterpart mapping dictionary constants `PEFT_CONFIG_MAPPING` and `ADAPTERS_CONFIG_MAPPING`. If the PEFT method in the configuration is contained within the mapping dictionaries, a specific config is used. Otherwise, it will default to `PeftArguments` (in file `hparams/parser.py`, function `_parse_train_args`). Importantly, every PEFT method, whether added from the Adapters

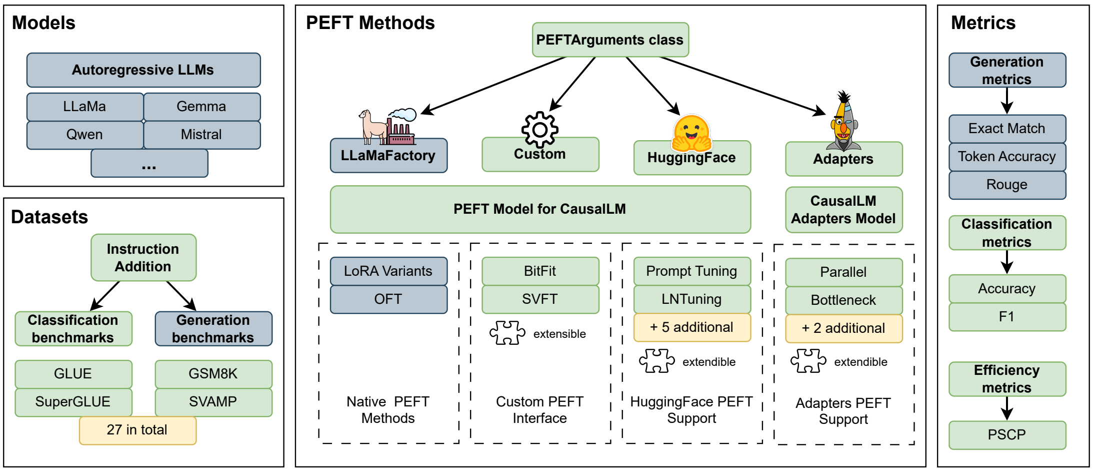

Figure 1: Diagram representing the components of PEFT-FACTORY. The four main overarching components of PEFT-FACTORY are PEFT Methods, Datasets, Models, and Metrics, which are further defined by their subcomponents. Components represented by green color are implemented in PEFT-FACTORY, components in blue color are native to LLaMA-Factory (Zheng et al., 2024a). Additionally, the Adapters library requires a different model class than the rest of the PEFT provider frameworks.

library or the HuggingFace PEFT library, comes with its own set of parameters or hyperparameters that can be tuned. PEFT-FACTORY automatically detects the parameters required by a specific PEFT method and uses the `HFArgumentParser` to parse them from config files (if parameters are not specified in the config, default values are used from the original implementation). This allows easy configuration and hyperparameter tuning via a YAML config file or the Gradio user interface.

The newly created and parsed `PeftArguments` are then forwarded to the `model loader`, where the `init_adapter` method creates the PEFT model (in file `model/model_utils/adapter.py`, functions `_setup_custom_peft`, `_setup_adapters_peft` and `_setup_hf_peft`). From this part, we leave the loading, training, and model saving on PEFT libraries and the LLaMA-Factory framework.

3.2 Custom PEFT Interface

In contrast to LLMA-Factory, PEFT-FACTORY implements a dynamic loading mechanism for custom PEFT methods, ultimately enabling its extensibility and modularity. This design allows researchers and practitioners to seamlessly integrate custom PEFT implementations without modifying the core codebase of PEFT-FACTORY. To demonstrate our Custom PEFT Interface, we replicated 2 PEFT methods that are not a part of any off-the-shelf PEFT framework, namely BitFit (Ben Zaken et al., 2022) and SVFT (Lingam et al., 2024) (located in the `peft directory`).

During the process of dynamic loading, the `peft_loader` module automatically discovers and loads PEFT methods from a structured directory hierarchy (in file `extras/peft_loader.py`, function `discover_custom_peft_methods`). Each custom PEFT method is organized in its own subdirectory containing two essential components: a `config.py` file defining a `PeftConfig` subclass, and a `model.py` file implementing a `BaseTuner` subclass. The configuration and model subclasses need to inherit the `PeftConfig` dataclass and `BaseTuner` abstract class from the HuggingFace PEFT library. The methods required to be implemented are then specified by the `BaseTuner` description in `tuner_utils.py`.

The loader validates each implementation by checking for required attributes (`peft_type` for configurations and `prefix` for model classes) before registration. The loader dynamically loads the config and model subclasses, registering them via the `register_peft_method` function (in file `peft_loader.py`), which adds the config and model to the constants of the Hugging Face PEFT library. Additionally, this process is also defined by the Algorithm 1, which explains how dynamic loading is implemented.

After the dynamic loading, the corresponding custom PEFT method is handled similarly to off-the-shelf methods as described in Section 3.1. As a result, the ease of configuration and hyperparameter tuning for the newly added PEFT methods also remains unchanged.

Algorithm 1 Dynamic PEFT Method Discovery

Input: PEFT directory path D
Output: Dictionary M mapping method names to (config, model) tuples
$M \leftarrow \emptyset$
for each subdirectory d in D **do**
    if d contains `config.py` and `model.py` **then**
        Load config class C from `config.py`
        Load model class T from `model.py`
        if C validates and T validates **then**
            $M[\text{name}(d)] \leftarrow (C, T)$
        end if
    end if
end for

To add a new method, it is required to match the directory structure inside the *PEFT methods directory* (the directory can be specified by the environment variable `PEFT_DIR` with `./peft` as the default directory) to match the organizational structure and class inheritance. We provide information about example method templates in Appendix C as well as in the PEFT-FACTORY documentation[2].

This plugin-style architecture promotes code reusability and enables fast prototyping of novel PEFT methods. Researchers can develop and test new methods separately, with the framework automatically integrating them at runtime.

The dynamic loading approach has proven particularly valuable for comparative studies, allowing researchers to evaluate multiple PEFT variants under identical experimental conditions without code duplication or version control conflicts.

3.3 Improved Dataset Loader

Besides a native support of text generation tasks (inherited from the LLaMA-Factory), we add support for classification tasks (in case of autoregressive models, the classification task $Pr_\theta(y|X)$ is cast as a generation $Pr_\theta(Y|X)$ task). To this end, we adapt and include multiple classification benchmarks, including GLUE (Wang et al., 2018) and SuperGLUE (Wang et al., 2019). This was largely possible due to the dataset loading feature included in LLaMA-Factory. However, we needed to enhance the dataset loader with an additional parameter called *instruction*. This optional attribute was added to the *DatasetAttr* class (in file `data/processor/parser.py`), and during data preprocessing, the instruction is prepended to the input text for the LLM (in file `data/processor/converter.py`, class AlpacaDatasetConverter). This allows adding instructions that have dataset-specific formatting (e.g., ones recommended by the dataset authors)[3] or are designed for instruction tuning tasks.

Addition of classification datasets. From the GLUE benchmark, we include 8 classification datasets separated into 6 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* [4], *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).

From SuperGLUE, we include 7 datasets separated 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).

Addition of generation datasets. We also include generation datasets that are commonly used to benchmark generative LLMs. We cover 6 datasets for reasoning and natural language understanding separated 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 datasets for mathematical problem solving separated 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); and 3 datasets for code generation, namely *Conala* (Yin et al., 2018), *CodeAlpacaPy* (Chaudhary, 2023), and *APPS* (Hendrycks et al., 2021).

[2] For detailed information, please visit the Adding PEFT Methods section of the PEFT-FACTORY documentation.

[3] For detailed information, please visit the Adding Datasets section of the PEFT-FACTORY documentation.

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

Adapting and preprocessing datasets. Some datasets may require further (mostly minor) format changes to be compatible with the input formatting of PEFT-FACTORY. We further describe this preprocessing in the Appendix B.1.

3.4 Classification and Efficiency Metrics

PEFT-FACTORY also calculates classification and efficiency metrics during the prediction phase in addition to already existing token accuracy and semantic similarity metrics (i.e., Rouge (Lin, 2004) and Bleu (Papineni et al., 2002)). From the classification metrics, PEFT-FACTORY implements standard Accuracy and F1 metrics. To measure efficiency during the evaluation of PEFT methods, PEFT-FACTORY implements the PSCP metric (Belanec et al., 2025), a highly adjustable metric that considers various efficiency parameters (e.g., number of parameters, memory usage, inference time).

We implement these metrics within the `train/sft/metric.py` file for supervised fine-tuning, following the pattern from LLaMA-Factory and utilizing separate data classes for each metric. Specifically, we name the classes *ComputeAccuracy*, *ComputeF1*, and *ComputePSCP*. For classification, we include a binary flag attribute within the training arguments, called *compute_classification_metrics*, which enables or disables the computation of classification metrics. For the efficiency metrics, we include a binary flag *compute_pscp*. Additional information on the usage of efficiency metrics can be found in Appendix B.2.

4 PEFT-FACTORY-enabled Use Cases

The extensibility of PEFT methods and datasets, together with a ready-to-use, controlled, and stable environment, is a key factor of PEFT-FACTORY that aims to promote further research on PEFT methods. To demonstrate how PEFT-FACTORY improves reproducibility and benchmarking of PEFT methods, we present two specific use cases.

4.1 PEFT Methods Reproducibility

How the modular design of PEFT-FACTORY promotes reproducibility and transparency of newly created PEFT methods can be seen in the use case, when fellow researchers and practitioners develop new PEFT methods.

Currently, when a new PEFT method is developed, the published source code is often not fully functional or difficult to reproduce. In addition, the authors often have to implement code for training and evaluation of the PEFT method from scratch, which is often repetitive, increases the probability of mistakes in the code, and is prone to inconsistencies in the final results.

In our scenario, authors only need to create a minimum number of files that are directly and solely connected to the design of the PEFT method itself. If the authors maintain the structure compatible with the PEFT-FACTORY custom PEFT interface, they can simply share it within the PEFT methods directory, create a configuration for training and evaluation, and run experiments on vast amounts of datasets and autoregressive models. Additionally, if the authors choose to implement their method inside any of the supported PEFT provider frameworks (i.e., Hugging Face PEFT or Adapters), only a small change is needed to contribute it to the next version of the PEFT-FACTORY[5].

4.2 PEFT Methods Benchmarks

Another possible use case of PEFT-FACTORY is to benchmark PEFT methods. To this end, PEFT-Factory provides a standardized and reproducible environment that eliminates inconsistencies in experimental setups (e.g., different seeds, hyperparameters or dataset splits), allowing researchers to reliably compare PEFT methods under identical conditions. To illustrate the benchmarking capability, Table 2 provides results from fine-tuning three PEFT methods on four different datasets using the LLaMA-3.2-1B-Instruct (Dubey et al., 2024) autoregressive model. Even such a small demonstrative comparison would require significant codebase preparation to execute the experiments, which PEFT-FACTORY eliminates to a minimum. From this evaluation, we can see that BitFit achieves the highest results in most of the datasets.

As a more complex benchmarking use case, we refer to our parallel work (Belanec et al., 2025), in which we introduce the PEFT-Bench – a benchmark of the efficiency of PEFT methods fully conducted within PEFT-FACTORY. PEFT-Bench provides the first unified, end-to-end benchmarking suite for evaluating PEFT methods on modern autoregressive LLMs, covering 27 datasets, 12 task types, and 7 diverse PEFT techniques. This benchmark was only possible due to PEFT-FACTORY, which serves as the underlying engine.

[5]We provide information on how to request the addition of a new PEFT provider method in the Contributing page of PEFT-FACTORY documentation

Method	SST-2	CoLA	WSC	SVAMP
BitFit	**97.5**	86.9	**55.2**	**92.3**
IA^3	95.3	85.3	3.6	84.1
Prefix Tuning	96.3	**88.8**	0.8	91.4

Table 2: Macro F1 results to demonstrate the benchmarking use case of PEFT-FACTORY on different datasets for different PEFT methods.

PEFT-FACTORY thus allows the community to easily extend the PEFT-Bench with new PEFT methods or even design new benchmarks with minimal effort. By ensuring experiment equivalency, replicability, and ease of extensibility, PEFT-FACTORY empowers researchers and practitioners to rigorously evaluate existing PEFT approaches and accelerate the development of new ones.

5 Conclusion and Future Work

We introduce PEFT-FACTORY, a modular and extensible framework for fine-tuning modern autoregressive models using recent and diverse PEFT methods. PEFT-FACTORY not only provides a way to utilize PEFT methods but also implements support for various PEFT providers and a custom PEFT interface to promote replicability and transparency when designing new PEFT methods. When comparing PEFT-FACTORY to various popular fine-tuning frameworks, as well as to our upstream framework, LLaMA-Factory, its novelty lies in suppo rting different PEFT methods, classifying tasks with custom instructions, and providing PEFT- and dataset-level extensibility.

Sustainability and Maintenance. To keep up with the updates included in LLaMA-Factory (which often include support of new LLMs or improvements in the training pipeline), we will regularly release a new version of PEFT-FACTORY (this includes merging the upstream changes into our repository). Additionally, to include new features in PEFT-FACTORY itself, we will reguraly release a separate version increment. Each change will be documented in the changelog of the specific release.

As the next steps, we would like to increase support for additional PEFT off-the-shelf methods, as well as reproduce some popular PEFT methods that are not currently supported by any of the PEFT provider frameworks. We believe that PEFT-FACTORY is an important and enabling tool that will promote the research of PEFT methods and allow their fair and consistent evaluation.

Acknowledgments

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

Part of the research results was obtained using the computational resources procured in 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.

References

Abubakar Abid, Ali Abdalla, Ali Abid, Dawood Khan, Abdulrahman Alfozan, and James Zou. 2019. Gradio: Hassle-free sharing and testing of ml models in the wild. *arXiv preprint arXiv:1906.02569*.

Lightning AI. 2023. Litgpt. https://github.com/Lightning-AI/litgpt.

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.

Yuvanesh Anand, Zach Nussbaum, Brandon Duderstadt, Benjamin Schmidt, and Andriy Mulyar. 2023. Gpt4all: Training an assistant-style chatbot with large scale data distillation from gpt-3.5-turbo. https://github.com/nomic-ai/gpt4all.

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.

Axolotl maintainers and contributors. 2023. Axolotl: Open source llm post-training.

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, Branislav Pecher, Ivan Srba, and Maria Bielikova. 2025. Peft-bench: A parameter-efficient fine-tuning methods benchmark. *arXiv preprint*.

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.

Arthur Cayley. 1846. Sur quelques propriétés des déterminants gauches. *Journal für die reine und angewandte Mathematik*.

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.

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.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: efficient finetuning of quantized llms. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Shizhe Diao, Rui Pan, Hanze Dong, Ka Shun Shum, Jipeng Zhang, Wei Xiong, and Tong Zhang. 2023. Lmflow: An extensible toolkit for finetuning and inference of large foundation models. *arXiv preprint arXiv:2306.12420*.

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*.

Peng Gao, Jiaming Han, Renrui Zhang, Ziyi Lin, Shijie Geng, Aojun Zhou, Wei Zhang, Pan Lu, Conghui He, Xiangyu Yue, Hongsheng Li, and Yu Qiao. 2023. Llama-adapter v2: Parameter-efficient visual instruction model. *arXiv preprint arXiv:2304.15010*.

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.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *Transactions on Machine Learning Research*.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. 2024. Lora+: efficient low rank adaptation of large models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*.

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*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

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.

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

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.

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.

Shenggui Li, Hongxin Liu, Zhengda Bian, Jiarui Fang, Haichen Huang, Yuliang Liu, Boxiang Wang, and Yang You. 2023. Colossal-ai: A unified deep learning system for large-scale parallel training. In *Proceedings of the 52nd International Conference on Parallel Processing*, ICPP '23, page 766–775, New York, NY, USA. Association for Computing Machinery.

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*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

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.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: weight-decomposed low-rank adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

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. 2023. Gpt understands, too. *AI Open*.

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`.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. 2024. Pissa: principal singular values and singular vectors adaptation of large language models. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

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

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.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020. MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, 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.

Clifton Poth, Hannah Sterz, Indraneil Paul, Sukannya Purkayastha, Leon Engländer, Timo Imhof, Ivan Vulić, Sebastian Ruder, Iryna Gurevych, and Jonas Pfeiffer. 2023. Adapters: A unified library for parameter-efficient and modular transfer learning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 149–160, Singapore. Association for Computational Linguistics.

Zeju Qiu, Weiyang Liu, Haiwen Feng, Yuxuan Xue, Yao Feng, Zhen Liu, Dan Zhang, Adrian Weller, and Bernhard Schölkopf. 2023. Controlling text-to-image diffusion by orthogonal finetuning. *Advances in Neural Information Processing Systems*, 36:79320–79362.

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.

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.

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*.

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

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. 2020. Trl: Transformer reinforcement learning. `https://github.com/huggingface/trl`.

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*.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Raghavi Chandu, David Wadden, Kelsey MacMillan, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. 2023a. How far can camels go? exploring the state of instruction tuning on open resources. *Preprint*, arXiv:2306.04751.

Zhen Wang, Rameswar Panda, Leonid Karlinsky, Rogerio Feris, Huan Sun, and Yoon Kim. 2023b. Multitask prompt tuning enables parameter-efficient transfer learning. In *The Eleventh International Conference on Learning Representations*.

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.

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, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

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*.

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, and 25 others. 2024. Qwen2.5 technical report. *ArXiv*, abs/2412.15115.

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.

Renrui Zhang, Jiaming Han, Chris Liu, Peng Gao, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, and Yu Qiao. 2023. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*.

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*.

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

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. 2024b. Galore: memory-efficient llm training by gradient low-rank projection. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

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. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. 2024a. 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)*, pages 400–410, Bangkok, Thailand. Association for Computational Linguistics.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024b. 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 Ethical Considerations

The experiments in this paper were conducted using publicly available datasets, including SST-2, CoLA, WSC, and SVAMP, as cited by 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 went through 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. Additionally, we do not collect or utilize any personally identifiable information or offensive content, and we do not engage in crowdsourcing for data annotation in any form. To our knowledge, we are not aware of any potential ethical harms or negative societal impacts of our work, apart from those related to the field of Machine Learning (i.e., the use of computational resources that consume energy and produce heat, resulting in indirect CO2 emissions). We follow the license terms for the LLaMa-3.2-1B-Instruct model we use; all models and datasets permit 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.

Importantly, in line with the open-science spirit, PEFT-FACTORY is an open-source downstream fork of LLaMA-Factory, licensed under the Apache-2.0 license (we respect the license and add append headers to the files that we have added or modified).

Impact Statement: CO2 Emissions Related to Experiments. The experiments in this paper require GPU computing resources as we train and evaluate 1 model for different methods (3) and datasets (4). Overall, the experiments, including evaluations (which did not require training but still utilized GPU resources for inference) and preliminary experiments (which are outside the scope of our work), were conducted using a private infrastructure with a carbon efficiency of 0.432 kgCO_2eq/kWh. Approximately 50 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_2eq, of which 0% were directly offset. Whenever possible, we tried to reduce the computational costs.

B Further Details

In this section, we include detailed information about PEFT-FACTORY that can be used by advanced users to further understand or extend our framework. In Table 3, we provide a comparison of different easy-to-use fine-tuning frameworks in terms of available PEFT methods, highlighting the undeniable contribution of PEFT-FACTORY.

B.1 Preprocessing datasets

Out of 27 included datasets, we namely preprocess and adapt MultiRC, WiC, COPA, ReCoRD, WSC, MMLU, PIQA, SIQA, HellaSwag, Wingrande, OBQA, MathQA, and SVAMP datasets. We upload all our adapted and preprocessed dataset versions to HuggingFace Hub[6]. Additionally, some datasets contain numerical values by default, formatted as `class values` in the HuggingFace dataset class. We convert such formats to textual representations to ensure compatibility with autoregressive generative models. Therefore, we transform multiple input and output columns of a single dataset to just a two-column format, including only *input* and *output* for the LLM.

B.2 Efficiency Metrics

The PSCP metric comprises a set of constants that must be configured to function properly. Specifically *pspc_cp*, *pscp_cf*, *pscp_cm*, *pspc_bp*, *pscp_bf*, and *pscp_bm*. The C values are set by the first three attributes, and the β values are set by the last three attributes. We also provide default values for these attributes.

The C values in PSCP calculation represent reference constants used for scaling the parameters (pscp_cp), inference time (pscp_cf), and peak memory usage (pscp_cp). The β values are defaulty set to 1, but can be set to any positive number. The higher the number, the higher the importance of the number of parameters *pspc_bp*, inference time *pspc_bf*, and peak memory usage *pspc_bm*. For detailed information on how to set these constants and the full equation, see Belanec et al. (2025).

B.3 Graphical User Interface

PEFT-FACTORY utilized LLaMA-Board graphical user interface based on Gradio (Abid et al., 2019). In this section, we describe the changes to

[6] https://hf.co/collections/kinit/peft-factory

	Reparametrized								Soft Prompt-Based					Adapter-Based				Selective		PEFT Extensibility
	LoRA	QLoRA	DoRA	LoRA+	PiSSA	GaLore	OFT	SVFT	Prompt Tuning	Prefix Tuning	P-Tuning	P-Tuning V2	MTP	IA^3	Bottleneck Adapter	Sequential Bottleneck Adapter	Parallel Adapter	BitFit	LNTuning	
PEFT-FACTORY	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
LLaMA-Factory	✔	✔	✔	✔	✔	✔	✔													
FastChat	✔	✔				✔														
LitGPT	✔	✔																		
LMFlow	✔	✔				✔														
Axolotl	✔	✔																		
Open-Instruct	✔	✔				✔														
H2O LLM Studio	✔	✔	✔																	

Table 3: Comparison of different PEFT methods available in PEFT-FACTORY with popular LLM fine-tuning frameworks. Current frameworks do not include support for other than reparametrized PEFT methods, while most of them are LoRA variations. These are all PEFT methods that were tested for functionality. PEFT Extensibility means that the framework also supports the modular addition of newly created PEFT methods, either by PEFT provider frameworks or directly by users.

the graphical user interface that enable fine-tuning LLMs with various PEFT methods.

During construction of the Gradio interface, PEFT-FACTORY takes the available PEFT methods and their configurations and constructs an interface for each configuration. Figure 2 shows the available PEFT methods to choose from the list. Each PEFT method contains default values that will be set automatically. However, the configuration can be further specified by the detailed configuration shown in Figure 3, which displays the configuration options for the Prompt Tuning method (Lester et al., 2021).

C Custom PEFT Method Templates

We provide minimal templates for the model.py and config.py files to design a PEFT method compatible with the PEFT-FACTORY custom PEFT interface, as documented in our framework[7].

Additionally, we provide an example directory structure (shown in Figure 5) that can be used to ensure compatibility with dynamic loading of PEFT-FACTORY.

Custom Method Directory Structure

```
./peft
└── <CustomMethod>
    ├── model.py
    └── config.py
```

Figure 5: Example directory structure of custom PEFT interface used for dynamic loading of PEFT custom methods.

[7]For detailed information, please visit the Templates section of PEFT-FACTORY documentation.

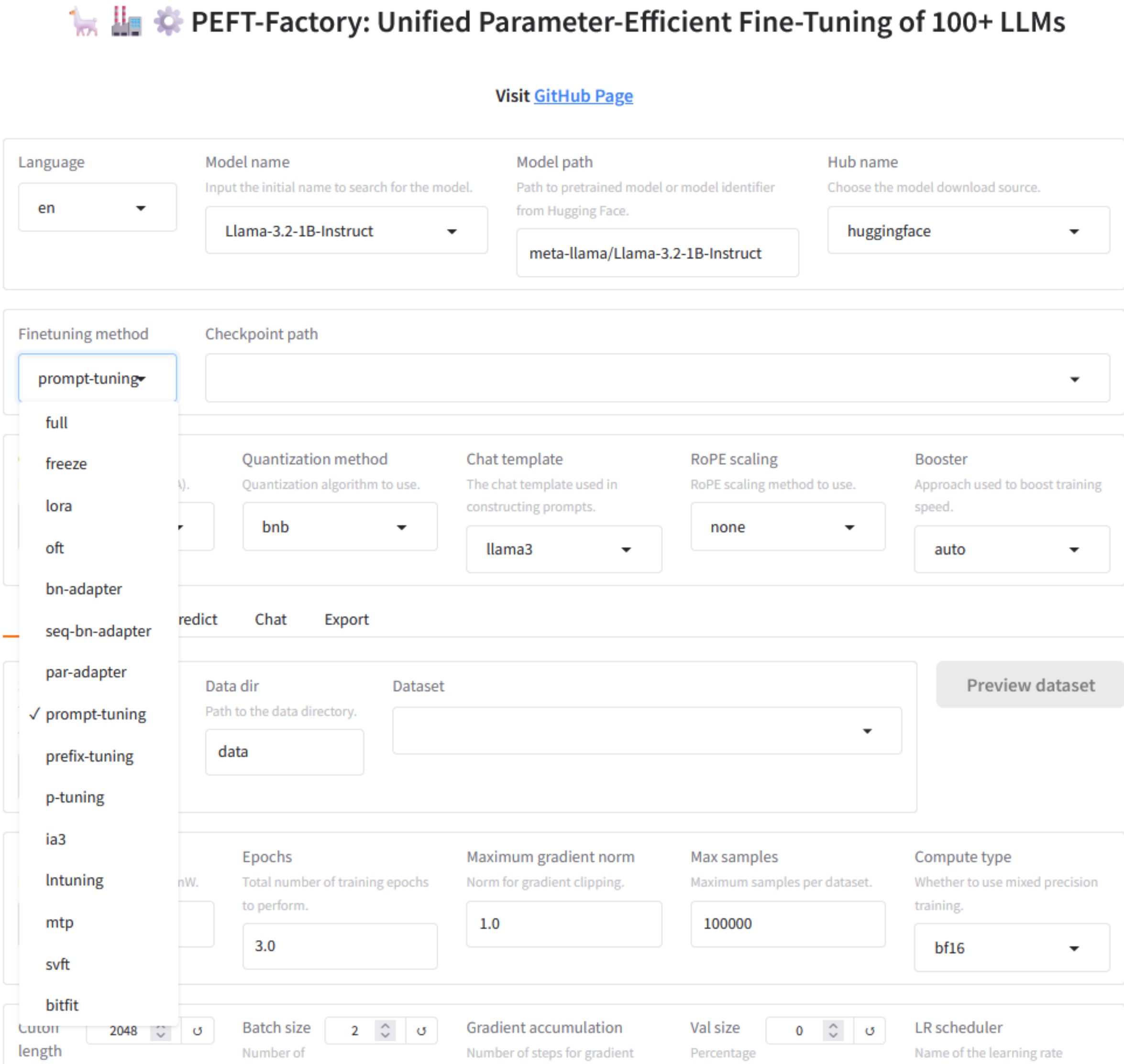

Figure 2: Selection of PEFT methods from `Finetuning method` dropdown menu. All 19 PEFT methods included in PEFT-FACTORY are available to choose.

Figure 3: Configuration options for the Prompt Tuning method.

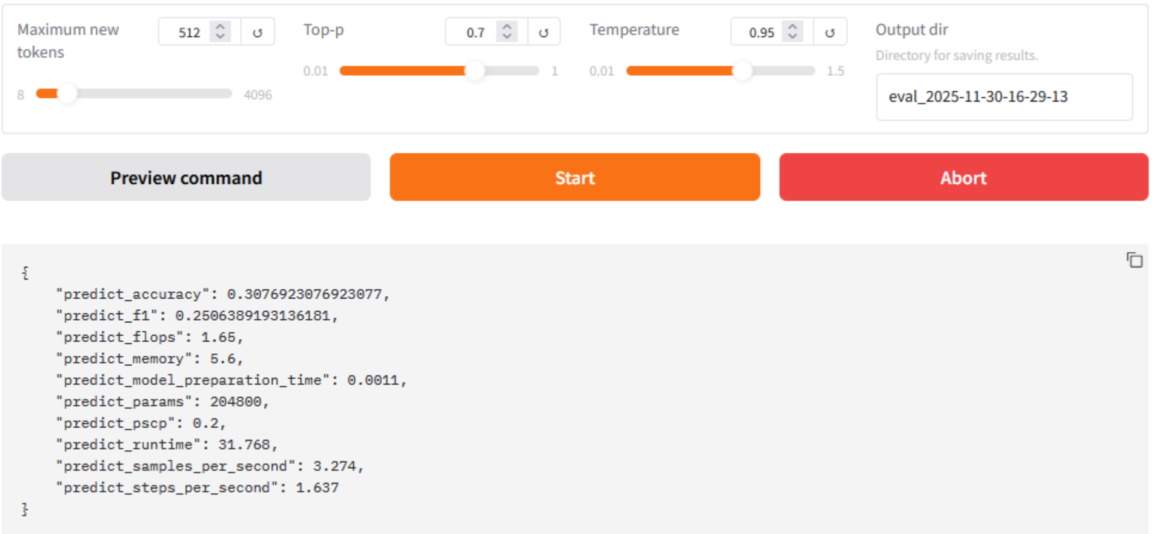

Figure 4: Classification and PSCP results for prediction after training with Prompt Tuning.