PORTLLM: PERSONALIZING EVOLVING LARGE LANGUAGE MODELS WITH TRAINING-FREE AND PORTABLE MODEL PATCHES

Rana Muhammad Shahroz Khan[1] **Pingzhi Li**[1*] **Sukwon Yun**[1*] **Zhenyu Wang**[1]
Shahriar Nirjon[1] **Chau-Wai Wong**[2] **Tianlong Chen**[1]
[1]The University of North Carolina at Chapel Hill [2]NC State University
* Denotes Equal Contribution

ABSTRACT

As large language models (LLMs) increasingly shape the AI landscape, fine-tuning pretrained models has become more popular than in the pre-LLM era for achieving optimal performance in domain-specific tasks. However, pretrained LLMs such as ChatGPT are periodically evolved (*i.e.*, model parameters are frequently updated), making it challenging for downstream users with limited resources to keep up with fine-tuning the newest LLMs for their domain application. Even though fine-tuning costs have nowadays been reduced thanks to the innovations of parameter-efficient fine-tuning such as LoRA, not all downstream users have adequate computing for *frequent personalization*. Moreover, access to fine-tuning datasets, particularly in sensitive domains such as healthcare, could be time-restrictive, making it crucial to retain the knowledge encoded in earlier fine-tuned rounds for future adaptation. In this paper, we present PORTLLM, a training-free framework that (*i*) creates an initial lightweight model update patch to capture domain-specific knowledge, and (*ii*) allows a subsequent seamless plugging for the continual personalization of evolved LLM at minimal cost. Our extensive experiments cover **seven** representative datasets, from easier question-answering tasks {BoolQ, SST2} to harder reasoning tasks {WinoGrande, GSM8K}, and models including {`Mistral-7B`, `Llama2`, `Llama3.1`, and `Gemma2`}, validating the portability of our designed model patches and showcasing the effectiveness of our proposed framework. For instance, PORTLLM achieves comparable performance to LoRA fine-tuning with reductions of up to $12.2\times$ in GPU memory usage. Finally, we provide theoretical justifications to understand the portability of our model update patches, which offers new insights into the theoretical dimension of LLMs' personalization.

1 INTRODUCTION

The rise of large pretrained language models has marked a significant shift in natural language processing (NLP), particularly in their ability to adapt to specific domains and tasks. These models, such as GPT-4 (Achiam et al., 2023), have achieved state-of-the-art performance by leveraging vast amounts of pretraining data (Antoniades et al., 2024). However, pretrained LLMs often require adaptation for specialized domains where context-specific knowledge is critical (Wang et al., 2022a; Bommasani et al., 2021; Qiu et al., 2020). Hence, while pretraining provides a strong foundation, *fine-tuning* (*e.g.*, *personalization*) is essential for specific domains. Fine-tuning bridges this gap by adapting pretrained models to specific tasks, enhancing their performance in domains such as healthcare, legal analysis, or scientific research (Min et al., 2021; Wei et al., 2021; Ouyang et al., 2022; Wang et al., 2022b; Liu et al., 2022; Raffel et al., 2020; Chen et al., 2024; Gao et al., 2024b). For instance, fine-tuned models can more effectively recognize domain-specific terminology, reason about complex relationships, and deliver more accurate and contextual appropriate responses.

Much effort has been devoted to developing fine-tuning methods. Traditionally, fine-tuning would normally involve updating all the parameters of a model. For example, in the case of Mistral 7B (Jiang et al., 2023), it would involve updating all 7 billion parameters. Typically, LLMs have

billions of parameters, and so this process poses significant challenges in computational and memory requirements. To alleviate these constraints, many Parameter Efficient Fine-Tuning (PEFT) methods (Houlsby et al., 2019) have been proposed. One popular method is Low-Rank Adaptation (LoRA) (Hu et al., 2021), which aims to estimate an updated matrix ΔW using the product of two low-rank matrices A and B. However, although LoRA lowers the training complexity, it still requires fine-tuning a large number of trainable parameters to reach a satisfactory performance. For example, a LoRA fine-tuning Llama 2 13B (Touvron et al., 2023) variant would require up to eight A6000 GPUs with 48 GB VRAM each, for a very small batch size, hence imposing a considerable memory and computational burden.

Furthermore, as cloud-hosted LLMs like ChatGPT (OpenAI, 2022) and Gemini (Team et al., 2023) undergo periodic (biannual or more frequent) updates with newer data, their improved performance often renders previous versions outdated. For downstream users who have already invested in fine-tuning these models for domain-specific tasks, repeatedly fine-tuning or performing personalization at every new update is highly impractical, as the process is not only computationally expensive but also time-consuming.

Beyond the computational and logistical hurdles, continual updates present another challenge for the downstream user: the lack of availability of the fine-tuning dataset. In domains such as healthcare or finance, data access is regulated by privacy laws and potentially time-sensitive. For example, fine-tuning medical models on patient data requires strict adherence to ethical and legal guidelines, making it difficult to repeatedly fine-tune with every model update. As a result, repeatedly fine-tuning models on newer LLM releases is not a viable long-term strategy for many users, hindering the downstream users from harnessing the performance gains from the evolving nature of LLMs. In response to these challenges, a natural research question comes:

> ***(Q)*** *How to leverage the personalized knowledge captured in the first fine-tuned model to update any evolving LLMs?*

To address this question, we introduce PORTLLM, a training-free framework that enables seamless transfer of domain-specific knowledge across evolving models. PORTLLM leverages model patches derived from LoRA, allowing users to port fine-tuned knowledge from one model iteration to another while preserving or even enhancing the performance of a downstream task. We show that our model patches are portable across model updates. If the downstream user fine-tunes a version of the model that has long become obsolete, they can simply add the model patches to the newer model, maintaining or boosting their performance on the downstream task. PORTLLM eliminates the need for costly periodic fine-tuning, offering a scalable solution for maintaining task-specific performance across different model versions. Our contributions can be summarized as follows:

❶ We introduce PORTLLM, a training-free framework designed to transfer knowledge between different versions of an evolving LLM. Given two model versions, PORTLLM leverages task-specific model patches extracted from a pretrained LLM and seamlessly applies them to the updated version of the pretrained LLM. This process allows the updated model to achieve comparable, and in some cases improved, performance on downstream tasks—without any need for fine-tuning.

❷ *Why do our model patches work?* We address this question through both theoretical analysis and empirical investigation. Our findings reveal that certain terms in the model patch are effectively negligible, enabling us to create a simplified version of the patch that requires no training. Consequently, adding the simplified model patches alone is sufficient to achieve improved performance on the downstream task. Furthermore, we examine the impact of the pretraining dataset on downstream tasks, demonstrating that our framework can harness the benefits of continued pretraining across different model updates and across different pretraining datasets including {OpenOrca, SlimOrca, OpenPlatypus, AlpacaGPT4}.

❸ We conduct extensive experiments across a series of **seven** downstream tasks, including Question-Answering Tasks {BoolQ, SST-2} (Wang, 2018; Wang et al., 2019), Similarity and Paraphrase Tasks {MRPC} (Wang, 2018), Inference Tasks {RTE, WNLI} (Wang, 2018), and Reasoning Tasks {WinoGrande, GSM8K} (Sakaguchi et al., 2021; Cobbe et al., 2021). To further demonstrate the robustness and broad applicability of our approach, we evaluate it on multiple model architectures, such as `Mistral-7B`, `Llama2-7B`, `Llama3.1-8B`, and `Gemma2-9B`. Additionally, we explore the applicability of our method on full-weight continued pretraining

compared to LoRA-based continued pretraining and show the effectiveness of our method by quantifying the gains on GPU memory usage (of up to $12.2\times$ reduction), GPU hours, and decrease in the number of trainable parameters (to zero trainable parameters).

2 Related Works

Large Language Models (LLMs). LLMs have transformed natural language processing, enabling models to perform complex tasks with remarkable accuracy and generalization. Models like GPT-3 (Brown et al., 2020), BERT (Devlin et al., 2019), and T5 (Raffel et al., 2023) have set benchmarks across a range of NLP tasks, from translation and summarization to question answering and text generation (Vaswani et al., 2023; Zhang et al., 2020; Rajpurkar et al., 2016). More recently, models like Llama (Touvron et al., 2023; Dubey et al., 2024), Mistral (Jiang et al., 2023), and Gemma (Team et al., 2024) have pushed the boundaries further by optimizing both performance and computational efficiency. LLama, Mistral, and Gemma represent recent advances in LLM architectures, each offering improvements in efficiency and performance. However, even with such improvements, the performance on domain-specific downstream tasks is sub-par making fine-tuning necessary. In this paper, we propose a training-free solution that enables the seamless transfer of personalized knowledge across evolving LLMs, reducing the need for costly fine-tuning and enhancing accessibility.

Parameter Efficient Fine-tuning (PEFT). The rapid growth in the size of pretrained LLMs has posed significant challenges for efficiently fine-tuning LLMs to specific downstream tasks. To address this, numerous PEFT methods have been developed, aiming to balance efficiency and accuracy. Early approaches focused on inserting trainable adapters—feed-forward networks placed between the layers of the pretrained model (Houlsby et al., 2019; Lin et al., 2020). Recent advancements have led to more sophisticated adapter-based PEFT methods (Mahabadi et al., 2021; Pfeiffer et al., 2020; Luo et al., 2023) including LMaaS (Sun et al., 2022) for service-oriented adaptation and kNN-Adapter (Huang et al., 2023) for retrieval-augmented fine-tuning. A notable example is LoRA (Hu et al., 2021), which introduces trainable low-rank weight perturbations to the pretrained model, significantly reducing the number of parameters required for fine-tuning. LoRA's key innovation lies in its use of the product of two low-rank matrices to approximate weight changes. Building upon this concept, several methods have emerged including Q-LoRA (Dettmers et al., 2023), CombLM (Ormazabal et al., 2023), and IPA(Lu et al., 2023). Concurrently, prompt-based learning methods have demonstrated effectiveness across various NLP tasks. Methods such as prompt-tuning (Lester et al., 2021), prefix-tuning (Li & Liang, 2021) and more recent approaches like Proxy-Tuning (Liu et al., 2024) and BBox-Adadpter (Sun et al., 2024) incorporate learnable continuous embeddings into the model's hidden states. They condition the frozen model to adapt to specific tasks without modifying the underlying architecture. Despite these advances, fine-tuning each updated LLM with PEFT to equip personalized knowledge remains highly costly, and how PEFT can bridge the gap in personalized settings within this evolving environment in a portable manner is yet to be fully explored. To this end, we develop in this paper the theory behind portable model patches that can be plugged into an updated model to carry over the personalized knowledge from the first fine-tuned model.

3 Methodology

3.1 Preliminaries

Transformers. Transformer models (Vaswani et al., 2023) is an architecture that has revolutionized NLP and other sequence-based tasks. It consists of two key components: (1) Multi-Head Self-Attention and (2) Feed-Forward Neural Network. The Multi-Head Self-Attention mechanism is the core innovation of transformers. It computes the weighted representation of the input sequence where each token attends to every other token. Suppose we are given an input $X \in \mathbb{R}^{n\times d}$ where n refers to the sequence length and d is the hidden dimension of the transformer model. Then for any given attention head i with a total of H heads, we define the following three matrices: query matrix $W_{q_i} \in \mathbb{R}^{d\times d_H}$, key matrix $W_{k_i} \in \mathbb{R}^{d\times d_H}$ and value matrix $W_{v_i} \in \mathbb{R}^{d\times d_H}$, where $d_H = d/H$. Given these, we can compute the self-attention for the ith head as follows

$$h_i = \text{Softmax}\left(\frac{XW_{q_i}(XW_{k_i})^T}{\sqrt{d_H}}XW_{v_i}\right), \quad i = 1, \ldots, H. \tag{1}$$

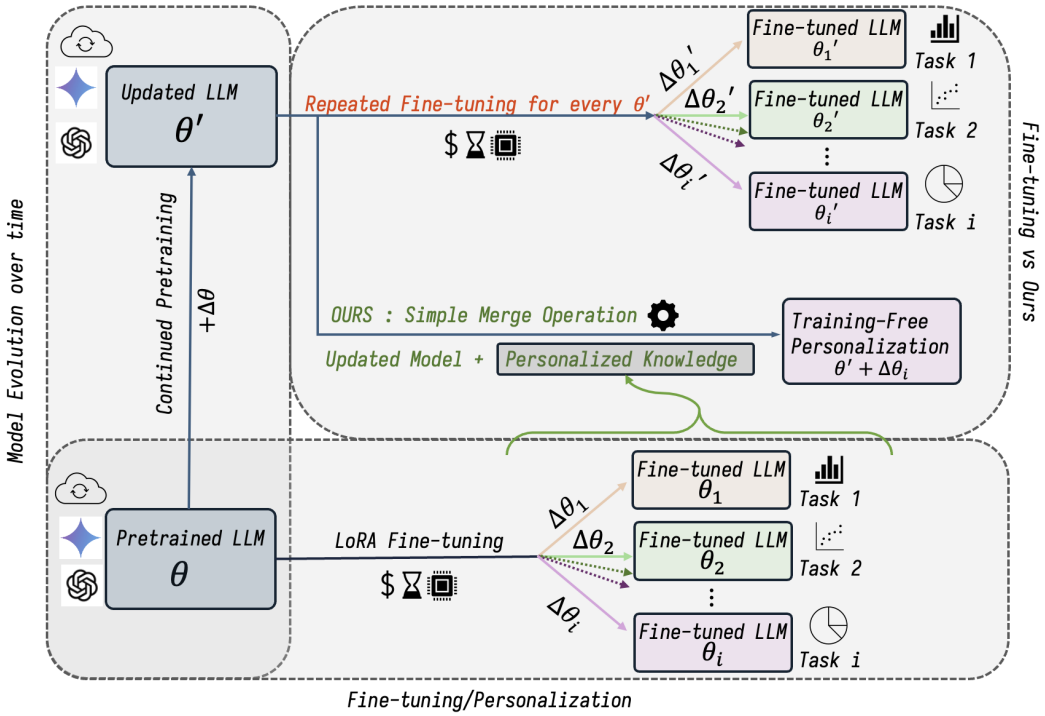

Figure 1: The diagram illustrates the core components of PORTLLM, a training-free framework to port personalized knowledge between evolving LLMs. Initially, a pretrained LLM is fine-tuned using LoRA. We transfer this task-specific knowledge without requiring the newer updated model to be fine-tuned again. This allows for continual performance improvements on downstream tasks without additional periodic fine-tuning.

The outputs of the H attention heads are then concatenated as follows:

$$\text{Multi-head Self-Attention}(X) = \text{Concatenate}(h_1, h_2, \ldots, h_H)W_o\,, \tag{2}$$

where $W_o \in \mathbb{R}^{d\times d}$ is a projection matrix that combines the outputs of the different heads back into the model's hidden dimension d. After the Muli-head Self-Attention, the output is passed through a position-wise feed-forward network that consists of two linear transformations and a nonlinear activation function (like ReLU or GELU). Given the weight matrix of the first linear layer $W_{\text{up}} \in \mathbb{R}^{d\times d_m}$, the weight matrix of the second layer $W_{\text{down}} \in \mathbb{R}^{d_m\times d}$, bias terms $b_1 \in \mathbb{R}^{d_m}$, $b_2 \in \mathbb{R}^{d}$, and a non-linear activation function $\sigma(\cdot)$ where d_m is the hidden dimension, Feed-Forward Network is applied independently to each position in the sequence as follows

$$\text{Feed-Forward Network}(X) = \sigma(XW_{\text{up}} + b_1)W_{\text{down}} + b_2\,. \tag{3}$$

Furthermore, these two layers are wrapped with residual connections and layer normalization.

Low-Rank Adaptation (LoRA). Consider a transformer model where $W_0 \in \mathbb{R}^{d\times d}$ is the pretrained weight matrix, which could be a weight matrix for any layer in the transformer. In a typical fine-tuning setup, the weights W_0 are updated during training to adapt the model to a specific task. This update requires storing and computing the entire matrix W_0 during training, which becomes computationally expensive for large models. Instead of updating the full weight matrix W_0, LoRA (Hu et al., 2021) assumes that the weight update $\Delta W \in \mathbb{R}^{d\times d}$, essentially the difference between pretrained weights W_0 and the hypothetical fine-tuned weight, can be approximated by a low-rank decomposition:

$$\Delta W = BA, \tag{4}$$

where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times d}$ are trainable matrices, while r is the rank of the decomposition with $r \ll d$. In this setup, the full weight update matrix ΔW is replaced by the product of two smaller matrices, B and A, drastically reducing the number of trainable parameters from d^2 to $2rd$. Hence, the fine-tuned model weights can be trained as follows:

$$W_{\text{new}} = W_0 + \Delta W = W_0 + BA. \tag{5}$$

By doing this, LoRA reduces the number of parameters that need to be trained while still allowing the model to adapt to new tasks.

3.2 Proposed Training-Free Framework: PortLLM

Notations and Assumptions. We refer to the pretrained LLM as the first version of the model, denoted by θ, and the updated continued pretrained model as θ', as illustrated in Figure 2. We also assume that to get to θ', the provider does continued pretraining using LoRA, where $\Delta\theta$ denotes this adapter, however empirical experiments in Section 4.4 show that even if the provider does full weight continued pretraining on the newer dataset, our method still holds. We denote this full weight updated model as ϕ. Similarly, for any downstream user i, we have a corresponding dataset denoted d_i. The base model fine-tuned on this dataset d_i is denoted by θ_i. We can also rewrite θ_i in terms of its LoRA update as $\theta_i = \theta + \Delta\theta_i$. Consequently, if we fine-tune updated model θ' for ith downstream task, we have $\theta'_i = \theta' + \Delta\theta'_i$. We further assume that a personalization adaptor $\Delta\theta_i$ or $\Delta\theta'_i$ has a rank that is much smaller than that of a continued pretrained adaptor $\Delta\theta$ or $\Delta\theta'$.

Proposed Method of PortLLM. PortLLM aims to approximate the fine-tuned updated model θ'_i by applying the older personalization adaptor/model patch $\Delta\theta_i$ to the continued pretrained model θ'. It will be shown in Section 3.3 that the older model patch $\Delta\theta_i$ may be used in lieu of the newer model patch $\Delta\theta'_i$, namely,

$$\theta'_i = \theta' + \Delta\theta'_i \approx \theta' + \Delta\theta_i. \tag{6}$$

In other words, one can readily add the extra knowledge $\Delta\theta_i$ from the previous fine-tuning process to the newest LLM θ'. Throughout our experimental section, we perform experiments with this approximated patching process.

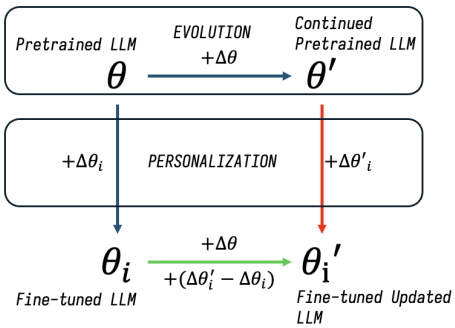

Figure 2: LLM's evolution & personalization cycle.

3.3 Analysis of Our Proposed Portability

Theoretical Justification. The fine-tuned updated model θ'_i can be decomposed into a naive update term and a residual matrix R as follows:

$$\theta'_i = \theta' + \Delta\theta'_i \tag{7a}$$

$$= \underbrace{(\theta' + \Delta\theta_i)}_{\text{Naive Update } \hat{\theta}'_i} + \underbrace{(\Delta\theta'_i - \Delta\theta_i)}_{\text{Residual Matrix } R}. \tag{7b}$$

Lemma 1 (informal): We claim that the residual matrix R is negligible compared to the naive update $\hat{\theta}'_i$. The key reason is that the model patches $\Delta\theta'_i$ and $\Delta\theta_i$ are both low rank (recall the assumption in Section 3.2), whereas the pretrained models θ and θ' are primarily full rank matrices. We provide a formal proof in Appendix C.

Empirical Validation. We empirically show that the difference between two personalization updates, $R = \Delta\theta'_i - \Delta\theta_i$, is negligible when compared to the naive update term, $\theta' + \Delta\theta_i$. We use Frobenius norm, $\|\cdot\|_F$, and the maximum singular value, $\sigma_{\max}$, to measure the magnitudes of matrices. The results are summarized in Table 1 across four different downstream tasks {BoolQ, MRPC, RTE, WNLI}. We can see that the $\sigma_{\max}$ for first term are on average $66\times$ and the $\|\cdot\|_F$ is $136\times$ bigger in the favor of the first term, which implies that C is comparatively negligible, hence our model patch can be simplified as shown in (6).

Table 1: Comparison of the terms making up our framework across different datasets.

Term		BoolQ	MRPC	RTE	WNLI
$\theta' + \Delta\theta_i$	$\sigma_{\max}$	7.37	7.37	7.37	7.37
	$\|\cdot\|_F$	16.80	16.80	16.81	16.81
$\Delta\theta'_i - \Delta\theta_i$	$\sigma_{\max}$	0.19	0.14	0.10	0.08
	$\|\cdot\|_F$	0.21	0.13	0.12	0.09
$\sigma_{\max}/\sigma_{\max}$		38.77	51.37	76.32	96.24
$\|\cdot\|_F/\|\cdot\|_F$		79.04	126.70	145.43	194.30

Table 2: Zero-shot performance comparison of model patches and baselines models on `Mistral-7B`, using the *OpenOrca* dataset for continued pretraining.

Model Version	**BoolQ** Accuracy	**SST-2** Accuracy	**MRPC** Accuracy/F1	**RTE** Accuracy	**WinoGrande** Accuracy	**WNLI** Accuracy	**GSM8K** Accuracy
Pretrained LLM θ	83.58	66.86	65.20 / 73.70	67.51	74.11	57.76	6.37
Updated LLM θ'	87.46	82.91	74.75 / 83.73	75.09	75.06	57.72	15.16
Fine-tuned LLM θ_i	91.01	95.99	89.46 / 92.62	87.73	85.95	83.11	34.04
Fine-tuned Updated LLM θ'_i	90.67	96.22	89.20 / 93.03	89.89	86.05	82.08	34.95
$\theta' + \Delta\theta_i$ (Ours)	90.24	96.10	88.73 / 92.10	89.17	85.01	83.10	41.32

4 EXPERIMENTS

Datasets and Architecture. We evaluate our framework on a diverse set of datasets to demonstrate PORTLLM universality and effectiveness across various downstream tasks and domains. Specifically, we leverage datasets from the *GLUE* (Wang, 2018), *SuperGLUE* (Wang et al., 2019), *WinoGrande* (Sakaguchi et al., 2021) and *GSM8K* (Cobbe et al., 2021) benchmarks commonly used for such evaluation in literature. For Question Answering tasks, we utilize *BoolQ* (from *SuperGLUE*) and *SST-2* (from GLUE); for Similarity and Paraphrase Tasks, the *MRPC* (from *GLUE*) dataset; for Inference Tasks, the *RTE* and *WNLI* (both from *GLUE*) datasets; and lastly for Reasoning Tasks, we employ *WinoGrande* and *GSM8K*. This broad spectrum of tasks enables a comprehensive evaluation of our model's performance across diverse downstream applications. For continued pretraining datasets, we use the following: *OpenOrca* (Lian et al., 2023a), *SlimOrca* (Lian et al., 2023b), *OpenPlatypus* (Lee et al., 2023) and *AlpacaGPT4* (Peng et al., 2023). Additionally, we conduct extensive experiments across multiple model architectures to demonstrate the generalizability of our framework. Specifically, we test our method on `Mistral-7B` (Jiang et al., 2023), `Llama2-7B` (Touvron et al., 2023), `Llama3.1-8B` (Dubey et al., 2024), and `Gemma2-9B` (Team et al., 2024), showcasing the robustness and adaptability of our approach across different LLMs.

Training Details. To simulate the progression of time, we employ continued pretraining, where we transition from θ to θ' by taking a pretrained model and further pretraining it on a specific dataset using LoRA (Hu et al., 2021). In all continued pretraining scenarios, we maintain a constant rank $r = 64$ and $\alpha = 128$ with a learning rate of 0.0001 and 4 epochs. For downstream tasks, we also use LoRA, but in this case, we set the rank $r = 8$ consistently to ensure a fair comparison across tasks. Furthermore, for all LoRA applications, we optimize all the attention layers (Key, Value, Query, Projection) and all Feed-Forward Network layers (Up Projection, Down Projection, and Gate Projection, where applicable). For each downstream fine-tuning, we use a constant learning rate of 0.0004 while the number of epochs for each dataset $\{$*BoolQ, SST-2, MRPC, RTE, WinoGrande, WNLI, GSM8K*$\}$ is as follows $\{5, 5, 5, 5, 3, 5, 1\}$. Lastly, for fine-tuning on a specific downstream dataset mentioned, we solely use the train split and evaluate on the test split.

Evaluation Metrics. We use the *Language Model Evaluation Harness* (Gao et al., 2024a) by EleutherAI to assess the performance of all trained and fine-tuned models across the datasets in our experiments. All evaluations are conducted in a *zero-shot* setting rather than a few-shot setting. For datasets $\{$*BoolQ, SST-2, RTE, WinoGrande, WNLI*$\}$, we employ accuracy as the primary evaluation metric. In the case of $\{$*MRPC*$\}$, we utilize both accuracy and F1 score to provide a more comprehensive evaluation. For $\{$*GSM8K*$\}$, we had two evaluation options: (1) Flexible Match Accuracy or (2) Exact Match Accuracy. Due to the poor zero-shot performance of the models on *GSM8K* for exact matches, we opted for Flexible Match Accuracy.

4.1 SUPERIORITY OF PORTLLM FRAMEWORK

In this section, we compare the performance of our model patches against several baseline models, including the pretrained LLM θ, the updated model using continued pretraining θ', the fine-tuned model θ_i, and the updated fine-tuned model θ'_i. For consistency, all models are variations of `Mistral-7B`, while the continued pretraining dataset is *OpenOrca*. Importantly, the performance

Table 3: Performance comparison of model patches $\Delta\theta_i$ across four downstream tasks {*BoolQ, MRPC, WNLI, WinoGrande*} using different continued pretraining datasets {*OpenOrca, SlimOrca, OpenPlatypus, AlpacaGPT4*}. All models are based on `Mistral-7B`.

Dataset	**Model**	**BoolQ** Accuracy	**MRPC** Accuracy	**WNLI** Accuracy	**WinoGrande** Accuracy
OpenOrca	Updated Model θ'	87.46	74.75	57.72	75.06
	Ours $\theta' + \Delta\theta_i$	90.24 (↑ 2.78)	88.73 (↑ 13.98)	83.10 (↑ 25.38)	85.01 (↑ 9.95)
SlimOrca	Updated Model θ'	87.16	74.76	64.79	74.98
	Ours $\theta' + \Delta\theta_i$	90.07 (↑ 2.91)	87.75 (↑ 12.99)	83.09 (↑ 18.30)	85.59 (↑ 10.61)
OpenPlatypus	Updated Model θ'	83.73	70.10	53.52	73.95
	Ours $\theta' + \Delta\theta_i$	90.34 (↑ 6.61)	90.20 (↑ 20.10)	80.28 (↑ 26.76)	83.58 (↑ 9.63)
AlpacaGPT4	Updated Model θ'	83.94	71.32	56.34	74.82
	Ours $\theta' + \Delta\theta_i$	90.52 (↑ 6.58)	89.22 (↑ 17.90)	84.51 (↑ 28.17)	84.93 (↑ 10.11)

is evaluated under the zero-shot setting across all the datasets. The results are summarized in Table 2.

❶ Compared to the zero-shot accuracy of updated model θ', applying our model patches can result in significant performance gains, with improvements up to $2.7\times$. Notably, no additional training is required when applying these patches, as the process simply involves a merge operation. Across all evaluated datasets {BooLQ, SST-2, MRCP, RTE, WinoGrande, WNLI, GSM8K}, we observe substantial zero-shot performance gains of $\{2.7\%, 13.19\%, 13.98\%, 14.08\%, 9.95\%, 25.38\%, 26.16\%\}$, respectively. This implies that our model patches are capable of transferring personalized knowledge across the different model versions.

❷ Comparing the performance of our model patches applied to the updated model $\theta' + \Delta\theta_i$ with that of the fine-tuned model θ_i, we observe that our method successfully transfers most of the downstream task-specific knowledge, yielding comparable results. As shown in Table 2, for tasks {*BoolQ*, *MRPC*, *WinoGrande*, *WNLI*}, the performance is nearly identical, with a maximum difference of only 0.77% in favor of the fine-tuned model. However, in tasks like {*SST-2*, *RTE*, *GSM8K*}, our approach outperforms the fine-tuned model by $\{0.11\%, 1.44\%, 7.28\%\}$, respectively. These results suggest that, when paired with a pretraining dataset that enhances performance for a specific task, our model patches can further leverage this advantage to improve downstream task performance in certain cases.

❸ Moreover, our method performs on par with the fine-tuned updated model ($\theta' + \Delta\theta_i$ compared to θ'_i), as evident from the comparison of the last two rows in Table 2. The difference between the two approaches is minimal when it comes to performance, with a maximum variation of just 1.04% observed in the case of *WinoGrande*. Notably, while one method requires fine-tuning, our approach remains completely training-free. Additionally, for certain downstream tasks such as *WNLI* and *GSM8K*, our method outperforms the fine-tuned updated model by 1.02% and 6.37%, respectively. This demonstrates that our approach not only provides comparable results but can, in some instances, surpass the performance of a fine-tuned evolved model.

4.2 Consistent Results across Different Pretraining Datasets

In this section, we investigate the impact of different pretraining datasets on our model patches $\Delta\theta_i$ and assess whether our method can effectively leverage updates obtained through continued pretraining. We conduct a comparative analysis across four downstream datasets – {*BoolQ, MRPC, WNLI, WinoGrande*} – alongside four distinct continued pretraining datasets: *OpenOrca*, *SlimOrca*, *OpenPLatypus*, and *AlpacaGPT4*. Consistent with our previous section, all experiments utilize the `Mistral-7B` model. The results of this analysis are summarized in Table 3.

❶ The results presented in Table 3 demonstrate that our model patches exhibit strong portability across different downstream tasks, irrespective of the continued pretraining dataset used. For each specific downstream task, we observe substantial improvements in zero-shot performance compared to the updated model θ' across all pretraining datasets. For instance, in the case of *WNLI*, the perfor-

Table 4: Performance analysis of model patches across various architectures {`Mistral-7B`, `Llama2-7B`, `Llama3.1-8B`, `Gemma2-9B`} on four downstream tasks {*BoolQ, MRPC, WNLI, WinoGrande*} under four different model settings. For each downstream task Cyan highlights the best performance for each model architecture.

Model	**Version**	**BoolQ** Accuracy	**MRPC** Accuracy/F1	**WNLI** Accuracy	**WinoGrande** Accuracy
Mistral 7B	Pretrained Model θ	83.58	65.20 / 73.70	57.76	74.11
	Updated Model θ'	87.46	74.75 / 83.73	57.72	75.06
	Fine-tuned Model θ_i	91.01	89.46 / 92.62	83.11	85.95
	Ours $\theta' + \Delta\theta_i$	90.24	88.73 / 92.10	83.10	85.01
Llama 2 7B	Pretrained Model θ	77.74	69.12 / 81.52	45.07	69.06
	Updated Model θ'	82.69	69.61 / 81.60	47.89	70.48
	Fine-tuned Model θ_i	88.21	88.97 / 92.01	57.75	75.45
	Ours $\theta' + \Delta\theta_i$	88.32	89.95 / 92.56	53.77	76.87
Llama 3.1 8B	Pretrained Model θ	82.35	66.91 / 77.54	59.15	73.56
	Updated Model θ'	85.63	75.00 / 83.60	61.95	73.64
	Fine-tuned Model θ_i	90.22	84.31 / 89.51	83.10	85.71
	Ours $\theta' + \Delta\theta_i$	90.03	89.71 / 92.71	81.69	84.85
Gemma 2 9B	Pretrained Model θ	83.98	68.63 / 74.19	57.75	74.19
	Updated Model θ'	88.41	77.21 / 84.93	74.65	76.72
	Fine-tuned Model θ_i	91.38	91.42 / 93.83	83.20	83.98
	Ours $\theta' + \Delta\theta_i$	91.16	90.69 / 93.17	88.73	83.82

mance boosts are $\{25.38\%, 18.30\%, 26.76\%, 28.17\%\}$ for {*OpenOrca, SlimOrca, OpenPlatypus, AlpacaGPT4*} datasets, respectively. A similar trend of significant improvement is also evident across other downstream tasks.

❷ We further observe from Table 3 that certain pretraining datasets can either enhance or detract from the zero-shot performance of θ' on specific downstream tasks, and this effect carries over to our frameworks to some degree. For instance, continued pretraining on *OpenPlatypus* leads to a decrease in performance on the *WNLI* dataset. Consequently, the addition of our model patch in this scenario results in the lowest accuracy for this particular downstream task among all the pretraining datasets evaluated.

4.3 Consistent Results across Different Model Architectures

This section provides a comprehensive analysis of our model patches across various architectures to evaluate their performance. We examine four different model architectures: {`Mistral-7B`, `Llama2-7B`, `Llama3.1-8B`, `Gemma2-9B`}, assessing their effectiveness on four distinct downstream tasks {*BoolQ, MPRC, WNLI, WinoGrande*}. Performance is evaluated under four settings : (1) pretrained model, (2) continued pretrained model or Updated Model, (3) fine-tuned model, and (4) our model patches ported to the updated model. The *OpenOrca* dataset is used for our continued pretraining in this analysis. The results are summarized in Table 4

❶ The results across different model architectures indicate that our training-free model patches significantly enhance zero-shot performance for all downstream tasks. In each case, our approach matches personalized performance (fine-tuned model), and in certain instances, it even surpasses it. For example, with `Gemma2-9B`, when the personalized performance exceeds our method, the difference in accuracy is at most 0.73%, which can be considered negligible. Conversely, in scenarios where our method outperforms personalized performance, we observe improvements of up to 5.53%. A similar trend is noted across the other model architectures, as detailed in Table 4.

❷ Additionally, we observe that fine-tuning is essential for achieving optimal zero-shot performance on downstream tasks. Across all model architectures, the zero-shot performance of both the pre-

Table 5: Evaluation of model patches added to `Mistral-7B` with full weight continued pretraining on the OpenOrca dataset across various downstream tasks.

Model Version	**BoolQ** Accuracy	**SST-2** Accuracy	**MRPC** Accuracy/F1	**RTE** Accuracy	**WinoGrande** Accuracy	**WNLI** Accuracy	**GSM8K** Accuracy
Full Weight Updated Model ϕ	86.61	93.81	77.21 / 85.31	75.09	72.77	63.38	20.55
$\phi + \Delta\theta_i$ (Ours)	89.88	95.53	87.25 / 90.97	90.61	85.08	80.28	41.24

trained model θ and the updated model θ' is subpar, with particularly poor results noted on tasks like *WNLI*. This reinforces the notion that excellent performance necessitates some form of fine-tuning, further motivating the need for our training-free framework. Another significant finding is the slight performance improvement for downstream tasks across all model architectures due to continued pretraining. Therefore, it is advisable for the downstream user to utilize the updated model weights, as this may provide beneficial enhancements in performance.

4.4 PortLLM Also Works with Full Weight Continued Pretraining

For our theoretical analysis, we initially assumed that continued pretraining was conducted using LoRA. However, we aim to investigate whether our method is effective across model evolution when the updates occur through full weight continued pretraining. Such a model is denoted ϕ. To evaluate this, we utilize `Mistral-7B`, which has undergone full weight continued pretraining on the OpenOrca dataset, and incorporate our model patches for various downstream tasks. We then compare the performance of these patched models against the zero-shot performance of ϕ to assess the improvements attributable to our model patches. The results across various downstream tasks are summarized in Table 5.

We find that our model patches can be effectively applied to a continued pretrained model utilizing full weight updates rather than relying solely on LoRA. Across all evaluated datasets – $\{$*BoolQ, SST-2, MRPC, RTE, WinoGrande, WNLI, GSM8K*$\}$ – we observe significant performance improvements of $\{3.27\%, 1.72\%, 10.04\%, 15.52\%, 12.31\%, 16.90\%, 20.69\%\}$, respectively, compared to the zero-shot performance of the Updated Model.

4.5 Computing Efficiency Comparison of PortLLM

This section evaluates the performance of our method from an efficiency perspective. We employ the following metrics for comparison: (1) Number of Trainable Parameters, (2) GPU Memory Utilization, and (3) GPU Hours. We analyze the merging of our model patches in relation to model fine-tuning using LoRA to achieve comparable performance. For LoRA fine-tuning calculations, we have the following settings: Downstream task *SST-2* for `Mistral-7B`, with local Batch Size of 4 and 5 Epochs. The results are summarized in Table 6.

Table 6: Efficiency Comparison Between PortLLM and LoRA on *SST-2* with `Mistral-7B` as the model architecture. The table compares trainable parameters, GPU Memory Usage, and GPU Hours for PortLLM and LoRA fine-tuning.

Metric	Ours $\theta' + \Delta\theta_i$	Fine-tuning Model θ_i	Savings
# Trainable Parameters	0	$20,971,520$	100%
GPU Memory Utilization (GB)	28.71	350.61	$12.21\times$
GPU Hours	0.0083	40.65	$4897\times$

Compared to downstream fine-tuning using LoRA, our method offers a plug-and-play solution with no trainable parameters, resulting in a reduction of nearly 20 million in parameters that need to be trained. This training-free paradigm not only conserves resources but also saves up to $12.2\times$ GPU Memory, reducing the requirement from 350 GB for LoRA to just 28.7 GB during the merge operation of model patches. Additionally, the merge operation can be executed in mere seconds, in contrast to the hours required for fine-tuning. This opens many other doors for applications of our model patches. PortLLM demonstrates the potential for on-device, training-free models for various down-stream tasks without the need for fine-tuning. Furthermore, it reduces the need for expensive cloud infrastructure, especially in large-scale fine-tuning.

5 CONCLUSION

In this paper, we propose PORTLLM, a framework aimed at addressing the challenges faced by downstream users of pretrained LLMs when adapting to frequent model evolutions over time. By leveraging lightweight model patches, PORTLLM offers a training-free, cost-effective solution to seamlessly transfer domain-specific knowledge between different iterations of LLMs. This enables users to maintain, and sometimes even enhance, their models' performance on specialized tasks without the need for repeated fine-tuning or extensive computational resources. Through extensive empirical evaluations across a set of tasks and models, we demonstrate that our method not only preserves performance but can also leverage the continual updates in pretraining LLMs, offering substantial gains in task-specific performance. Moreover, we provide theoretical insights into the portability of these model patches, highlighting the underlying factors that make them effective across evolving model versions. Looking forward, PORTLLM paves the way for more robust and adaptable solutions in the evolving landscape of LLM personalization, offering another avenue for training-free adaptation. Furthermore, future endeavours will aim at developing such methods that work across different model architecture, including using techniques from model merging.

6 REPRODUCIBILITY STATEMENT

To ensure reproducibility, we provide detailed descriptions of datasets, model architectures, training settings, and evaluation metrics used in our experiments in Section 4. The same section also dives deep into the hyperparameters used for all the tasks mentioned in our paper, including LoRA fine-tuning and downstream evaluation. Furthermore, we have also provided all the training scripts alongside the hyperparameters as supplementary material so that results from our papers can be reproduced with minimal effort. Lastly, the datasets and model architectures utilized in this paper are open-source and publicly available for anyone's use. Each dataset, as well as model, have been cited accordingly so that anyone can reproduce the experiments.

ACKNOWLEDGMENT

This research was, in part, funded by the CISCO Faculty Award, UNC SDS Seed Grant and NetMind.AI. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied of the funding organizations.

REFERENCES

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

Antonis Antoniades, Xinyi Wang, Yanai Elazar, Alfonso Amayuelas, Alon Albalak, Kexun Zhang, and William Yang Wang. Generalization vs memorization: Tracing language models' capabilities back to pretraining data. *arXiv preprint arXiv:2407.14985*, 2024.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, S. Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen A. Creel, Jared Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren E. Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas F. Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, O. Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir P. Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Benjamin Newman, Allen Nie,

Juan Carlos Niebles, Hamed Nilforoshan, J. F. Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Robert Reich, Hongyu Ren, Frieda Rong, Yusuf H. Roohani, Camilo Ruiz, Jack Ryan, Christopher R'e, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishna Parasuram Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei A. Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models. *ArXiv*, 2021. URL https://crfm.stanford.edu/assets/report.pdf.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. Graphwiz: An instruction-following language model for graph problems. *arXiv preprint arXiv:2402.16029*, 2024.

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

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient Finetuning of Quantized LLMs, 2023. URL https://arxiv.org/abs/2305.14314.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL https://arxiv.org/abs/1810.04805.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The LLaMA 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 2024a. URL https://zenodo.org/records/12608602.

Ziqi Gao, Xiangguo Sun, Zijing Liu, Yu Li, Hong Cheng, and Jia Li. Protein multimer structure prediction via prompt learning. *arXiv preprint arXiv:2402.18813*, 2024b.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-Efficient Transfer learning for NLP, 2019. URL https://arxiv.org/abs/1902.00751.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models, 2021. URL https://arxiv.org/abs/2106.09685.

Yangsibo Huang, Daogao Liu, Zexuan Zhong, Weijia Shi, and Yin Tat Lee. k nn-adapter: Efficient domain adaptation for black-box language models. *arXiv preprint arXiv:2302.10879*, 2023.

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

Ariel N. Lee, Cole J. Hunter, and Nataniel Ruiz. Platypus: Quick, Cheap, and Powerful Refinement of LLMs. 2023.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021. URL https://arxiv.org/abs/2104.08691.

Xiang Lisa Li and Percy Liang. Prefix-Tuning: Optimizing Continuous Prompts for Generation, 2021. URL https://arxiv.org/abs/2101.00190.

Wing Lian, Bleys Goodson, Eugene Pentland, Austin Cook, Chanvichet Vong, and "Teknium". OpenOrca: An Open Dataset of GPT Augmented FLAN Reasoning Traces. https://https://huggingface.co/Open-Orca/OpenOrca, 2023a.

Wing Lian, Guan Wang, Bleys Goodson, Eugene Pentland, Austin Cook, Chanvichet Vong, and "Teknium". SlimOrca: An Open Dataset of GPT-4 Augmented FLAN Reasoning Traces, with Verification, 2023b. URL https://https://huggingface.co/Open-Orca/SlimOrca.

Zhaojiang Lin, Andrea Madotto, and Pascale Fung. Exploring versatile generative language model via parameter-efficient transfer learning, 2020. URL https://arxiv.org/abs/2004.03829.

Alisa Liu, Xiaochuang Han, Yizhong Wang, Yulia Tsvetkov, Yejin Choi, and Noah A Smith. Tuning language models by proxy. *arXiv preprint arXiv:2401.08565*, 2024.

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

Ximing Lu, Faeze Brahman, Peter West, Jaehun Jung, Khyathi Chandu, Abhilasha Ravichander, Prithviraj Ammanabrolu, Liwei Jiang, Sahana Ramnath, Nouha Dziri, Jillian Fisher, Bill Lin, Skyler Hallinan, Lianhui Qin, Xiang Ren, Sean Welleck, and Yejin Choi. Inference-Time Policy Adapters (IPA): Tailoring Extreme-Scale LMs without Fine-tuning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Conference on Empirical Methods in Natural Language Processing*, pp. 6863–6883, Singapore, December 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.emnlp-main.424.

Gen Luo, Minglang Huang, Yiyi Zhou, Xiaoshuai Sun, Guannan Jiang, Zhiyu Wang, and Rongrong Ji. Towards efficient visual adaption via structural re-parameterization, 2023. URL https://arxiv.org/abs/2302.08106.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers, 2021. URL https://arxiv.org/abs/2106.04647.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. Metaicl: Learning to learn in context. *arXiv preprint arXiv:2110.15943*, 2021.

OpenAI. ChatGPT: Optimizing Language Models for Dialogue. *OpenAI Blog*, 2022. URL https://openai.com/research/chatgpt.

Aitor Ormazabal, Mikel Artetxe, and Eneko Agirre. CombLM: Adapting black-box language models through small fine-tuned models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Conference on Empirical Methods in Natural Language Processing*, pp. 2961–2974, Singapore, December 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.emnlp-main.180.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, 2022.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*, 2023.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. AdapterHub: A Framework for Adapting Transformers, 2020. URL https://arxiv.org/abs/2007.07779.

Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 63(10): 1872–1897, 2020.

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

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023. URL https://arxiv.org/abs/1910.10683.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text, 2016. URL https://arxiv.org/abs/1606.05250.

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

Haotian Sun, Yuchen Zhuang, Wei Wei, Chao Zhang, and Bo Dai. BBox-Adapter: Lightweight Adapting for Black-Box Large Language Models. *arXiv preprint arXiv:2402.08219*, 2024.

Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. Black-box tuning for language-model-as-a-service. In *International Conference on Machine Learning*, pp. 20841–20855. PMLR, 2022.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Alex Wang. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. SuperGlue: A stickier benchmark for general-purpose language understanding systems. *Advances in Neural Information Processing Systems*, 32, 2019.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-Instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*, 2022a.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ NLP tasks. *arXiv preprint arXiv:2204.07705*, 2022b.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization, 2020. URL https://arxiv.org/abs/1912.08777.

APPENDIX

A ANALYSIS OF MODEL PERFORMANCE UNDER MULTIPLE CONTINUAL UPDATES

To validate our framework's robustness under periodic updates, we conducted experiments simulating multiple rounds of continued pretraining. Using `Mistral-7B` as our base model, we performed four sequential updates using different pretraining datasets: OpenOrca $\rightarrow$ OpenPlatypus $\rightarrow$ Alpaca $\rightarrow$ GPT4-LLM-Cleaned. For the hyperparameters, we utilize the same settings described in Section 4. We evaluated our model patch on all seven downstream tasks after each update. The results are summarized in Table A. Our experiments yield the following key findings:

Time	**Model Version**	**BoolQ** Accuracy	**SST-2** Accuracy	**MRPC** Accuracy/F1	**RTE** Accuracy	**WinoGrande** Accuracy	**WNLI** Accuracy	**GSM8K** Accuracy
$T = 0$	Pretrained Model θ	83.58	66.86	65.20 / 73.70	67.51	74.11	57.76	6.37
None	Fine-tuned Model θ_i	91.01	95.99	89.46 / 92.62	87.73	85.95	83.11	34.04
$T = 1$	Updated Model θ'	87.46	82.91	74.75 / 83.73	75.09	75.06	57.72	15.16
OpenOrca	Ours $\theta' + \Delta\theta_i$	90.24	96.11	88.73 / 92.10	89.17	85.01	83.10	41.32
$T = 2$	Updated Model θ'	86.33	86.81	76.23 / 83.53	71.48	74.51	52.11	12.36
OpenPlatypus	Ours $\theta' + \Delta\theta_i$	89.88	96.22	88.24 / 91.55	88.09	84.37	83.10	42.15
$T = 3$:	Updated Model θ'	87.03	85.78	74.02 / 83.33	73.65	74.27	56.34	16.38
Alpaca	Ours $\theta' + \Delta\theta_i$	89.66	96.33	88.73 / 92.12	88.81	85.08	83.10	38.21
$T = 4$:	Updated Model θ'	85.11	75.79	72.78 / 82.79	74.37	71.67	56.34	14.94
GPT4-LLM	Ours $\theta' + \Delta\theta_i$	88.41	96.10	87.01 / 90.91	85.56	80.19	77.46	31.77

Table A: Model performance across sequential updates (T=0 to T=4) on seven downstream tasks. We report accuracy for all tasks except MRPC, where we show both accuracy and F1 score.

❶ Across all update stages ($T = 0$ to $T = 4$), applying our model patches results in substantial zero-shot improvements over the updated model $\theta\prime$. These improvements are consistent and significant across different tasks, with SST-2 showing gains from $+9.41\%$ to $+20.31\%$, WNLI maintaining strong improvements between $+21.12\%$ and $+30.99\%$, and MRPC consistently improving by $+12 - 14\%$ in accuracy. Notably, these improvements are achieved without any additional training, requiring only a simple merge operation of our model patches.

❷ The performance stability of our patched models is particularly noteworthy when compared to the fluctuating zero-shot performance of $\theta\prime$. As shown in Table A, our method maintains remarkably consistent performance across multiple updates. For instance, BoolQ accuracy remains within a tight range of $88.41 - 90.24\%$, SST-2 consistently maintains accuracy above 96%, and MRPC's F1 score stays above 90 across all update stages. These results demonstrate our method's robustness to successive model updates and its ability to preserve task-specific knowledge.

❸ Furthermore, our method shows interesting behavior in leveraging complementary knowledge from different updates. Taking GSM8K as an example, we observe varying but significant improvements ranging from $+16.83\%$ to $+29.79\%$ across different update stages. This suggests that our model patches can effectively combine knowledge from both the original fine-tuning and the continued pretraining updates, sometimes leading to performance gains that exceed what might be expected from either source alone. Such behavior demonstrates the potential of our approach to not just preserve but potentially enhance task performance through knowledge integration across model versions.

B ANALYSIS OF LORA RANK SELECTION FOR DOWNSTREAM TASKS

To validate our choice of LoRA rank and understand its impact on model performance, we conducted experiments with varying ranks. This analysis helps establish the optimal balance between computational efficiency and model effectiveness. To perform these experiments, we utilize `Mistral-7B` with OpenOrca as the continued pretraining dataset and evaluate on BoolQ, MRPC, WNLI downstream tasks with a varying rank for LoRA. The rest of the hyperparameters are the same as in Section 4. The results can be seen in Table B.

❶ Both fine-tuning θ_i and our method $\theta\prime + \Delta\theta_i$ demonstrate remarkable stability across different ranks, with optimal performance typically achieved at rank 8-16. Specifically, on BoolQ, we observe peak accuracy at rank 16, while MRPC and WNLI show optimal performance at rank 8. This consistency across ranks validates the robustness of our approach and suggests that model patches effectively capture task-specific knowledge regardless of rank selection.

❷ When examining the efficiency aspects, we find that increasing rank beyond 8 provides diminishing or even negative returns. As shown in Table B, at rank 32, we observe decreased performance across most tasks compared to rank 8: BoolQ drops by 1.01%, MRPC by 0.49%, and WNLI by 2.83%. Importantly, the performance gap between our method and direct fine-tuning remains minimal even at lower ranks, with differences of less than 1% in most cases. This suggests that our training-free approach maintains its effectiveness even with more constrained rank settings.

❸ Our analysis strongly justifies our original choice of rank 8 as the default setting. This configuration achieves an optimal balance between computational efficiency (requiring fewer parameters than higher ranks), model performance (maintaining competitive results across all tasks), and adaptation capability (providing sufficient capacity for task-specific learning). Notably, while higher ranks like 16 or 32 require significantly more parameters, they offer minimal or no performance benefits, making rank 8 the sweet spot for our training-free framework.

Rank	**Model Version**	**BoolQ** Accuracy	**MRPC** Accuracy/F1	**WNLI** Accuracy
$r = 2$	Fine-tuned Model θ_i	90.52	87.30 / 91.28	74.65
	Ours $\theta' + \Delta\theta_i$	89.85	87.01 / 91.03	77.46
$r = 4$	Fine-tuned Model θ_i	90.81	89.42 / 91.98	78.23
	Ours $\theta' + \Delta\theta_i$	90.18	87.75 / 91.53	80.28
$r = 8$	Fine-tuned Model θ_i	91.01	89.46 / 92.62	83.11
	Ours $\theta' + \Delta\theta_i$	90.24	88.73 / 92.10	83.10
$r = 16$	Fine-tuned Model θ_i	91.07	89.22 / 92.49	81.69
	Ours $\theta' + \Delta\theta_i$	90.98	88.97 / 92.31	82.98
$r = 32$	Fine-tuned Model θ_i	90.00	88.97 / 92.15	80.28
	Ours $\theta' + \Delta\theta_i$	90.28	88.73 / 91.93	81.69

Table B: Performance comparison across different LoRA ranks (2, 4, 8, 16, 32) on three downstream tasks using `Mistral-7B`.

C PROOF OF LEMMA 1

Notations: $\mathcal{C}(\cdot)$ returns the column vector subspace of a matrix.

Recall in Section 3.3, we decomposed the fine-tuned updated model θ'_i into two terms:

$$\theta'_i = \theta' + \Delta\theta'_i \tag{8a}$$

$$= \underbrace{(\theta' + \Delta\theta_i)}_{\text{Naive Update } \hat{\theta}'_i} + \underbrace{(\Delta\theta'_i - \Delta\theta_i)}_{\text{Residual Matrix } R}. \tag{8b}$$

Lemma 1: The residual matrix R is negligible compared to the naive update $\hat{\theta}'_i$ in terms of the Frobenius norm.

Proof: Our goal is to show that the error ratio $\|R\|_{\mathrm{F}}^2 \;/\; \left\|\hat{\theta}'_i\right\|_{\mathrm{F}}^2$ is small. We will proceed by finding a large enough numerator $\|R\|_{\mathrm{F}}^2$ and small enough $\left\|\hat{\theta}'_i\right\|_{\mathrm{F}}^2$ in the LoRA context and show that the upper bound of the error ratio is small.

Numerator $\|R\|_{\mathrm{F}}^2$. First, we search for conditions that potentially lead to residual matrices with larger Frobenius norm. We apply compact SVD to the patches $\Delta\theta_i$ and $\Delta\theta'_i$, designating subscripts

1 and 2 for SVD matrices, respectively:

$$\|R\|_{\mathrm{F}}^2 = \|\Delta\theta_i' - \Delta\theta_i\|_{\mathrm{F}}^2 \tag{9a}$$

$$= \left\|U_2\Sigma_2V_2^T - U_1\Sigma_1V_1^T\right\|_{\mathrm{F}}^2 \tag{9b}$$

$$= \left\|\sum_{\ell=1}^{r_2}\sigma_\ell^{(2)}u_\ell^{(2)}v_\ell^{(2)T} - \sum_{k=1}^{r_1}\sigma_k^{(1)}u_k^{(1)}v_k^{(1)T}\right\|_{\mathrm{F}}^2. \tag{9c}$$

Here, the typical order of magnitude for r_1 and r_2 is about 10.

1. When there is no intersection between the two pairs of singular vector subspaces, namely, $\mathcal{C}(U_1)\cap\mathcal{C}(U_2) = \varnothing$ and $\mathcal{C}(V_1)\cap\mathcal{C}(V_2) = \varnothing$, the two terms in (9c) may be combined to form a valid compact SVD of rank $r_1 + r_2$ as follows:

$$\|R\|_{\mathrm{F}}^2 = \left\|\sum_{\ell=1}^{r_2}\sigma_\ell^{(2)}u_\ell^{(2)}v_\ell^{(2)T} + \sum_{k=1}^{r_1}\sigma_k^{(1)}\cdot(-u_k^{(1)})v_k^{(1)T}\right\|_{\mathrm{F}}^2 \tag{10a}$$

$$= \left\|\sum_{k'=1}^{r_1+r_2}\sigma_{k'}^{(1,2)}u_{k'}v_{k'}^T\right\|_{\mathrm{F}}^2, \tag{10b}$$

where $\sigma_1^{(1,2)},\ldots,\sigma_{r_1+r_2}^{(1,2)}$ is a list of descending ordered positive numbers sampled without replacement from $\{\sigma_k^{(1)}\}_{k=1}^{r_1}$ and $\{\sigma_\ell^{(2)}\}_{\ell=1}^{r_2}$. Applying the Frobenius norm property to the SVD representation of a matrix, we obtain

$$\|R\|_{\mathrm{F}}^2 = \sum_{k'=1}^{r_1+r_2}\left[\sigma_{k'}^{(1,2)}\right]^2 = \sum_{\ell=1}^{r_2}\left[\sigma_\ell^{(2)}\right]^2 + \sum_{k=1}^{r_1}\left[\sigma_k^{(1)}\right]^2. \tag{11}$$

This is the case when the two patches $\Delta\theta_i$ and $\Delta\theta_i'$ contain only orthogonal information. This is not very realistic because the two patches were created on the same downstream task i that should lead to some information in common.

2. When the two patches have are oppositely embedded in one of the singular value subspaces, e.g., $U_2 = -U_1$ and $V_2 = V_1$, the two terms in (9c) can be merged and singular values with the same ranking will be summed up, namely,

$$\|R\|_{\mathrm{F}}^2 = \left\|\sum_{\ell=1}^{r_2}\sigma_\ell^{(2)}u_\ell^{(2)}v_\ell^{(2)} + \sum_{k=1}^{r_1}\sigma_k^{(1)}\cdot(-u_k^{(1)})v_k^{(1)T}\right\|_{\mathrm{F}}^2 \tag{12a}$$

$$= \left\|\sum_{\ell=1}^{r_2}\left[\sigma_\ell^{(2)} + \sigma_\ell^{(1)}\right]u_\ell^{(2)}v_\ell^{(2)}\right\|_{\mathrm{F}}^2 \tag{12b}$$

$$= \sum_{\ell=1}^{r_2}\left[\sigma_\ell^{(2)} + \sigma_\ell^{(1)}\right]^2, \tag{12c}$$

which can be easily shown that it is larger than the orthogonal case (11) due to the extra interaction term $\sum_{\ell=1}^{r_2}\sigma_\ell^{(2)}\sigma_\ell^{(1)}$. When $\sigma_\ell^{(2)} = \sigma_\ell^{(1)}$, this case corresponds to two patches having exactly opposite gradient update directions, which again, is not very realistic because the same downstream tasks are used to generate the update directions. Equations (11) and (12c) both correspond to extreme conditions, and the continuum in between should be more realistic. We will use the large numerator (12c) to examine the error ratio.

Denominator $\left\|\hat{\theta}_i'\right\|_{\mathrm{F}}^2$. We continue to apply compact SVD to the continued pretrained model θ' and patch $\Delta\theta_i$, designating subscripts 0 and 1 for SVD matrices, respectively:

$$\left\|\hat{\theta}_i'\right\|_{\mathrm{F}}^2 = \|\theta' + \Delta\theta_i\|_{\mathrm{F}}^2 \tag{13a}$$

$$= \left\|U_0\Sigma_0V_0^T + U_1\Sigma_1V_1^T\right\|_{\mathrm{F}}^2 \tag{13b}$$

$$= \left\|\sum_{j=1}^{r_0}\sigma_j^{(0)}u_j^{(0)}v_j^{(0)T} + \sum_{k=1}^{r_1}\sigma_k^{(1)}u_k^{(1)}v_k^{(1)T}\right\|_{\mathrm{F}}^2. \tag{13c}$$

Here, the typical order of magnitude for r_0 is about 100. 1. When there is no intersection between the two pairs of singular vector subspaces, the two terms may be combined to form a valid compact SVD of rank $r_0 + r_1$. Hence, similar to (11), we have

$$\left\|\hat{\theta}_i'\right\|_{\mathrm{F}}^2 = \sum_{j=1}^{r_0}\left[\sigma_j^{(0)}\right]^2 + \sum_{k=1}^{r_1}\left[\sigma_k^{(1)}\right]^2. \tag{14}$$

2. When the basis vectors of singular matrices (with one matrix having opposite signs) of the patch can be found in the singular matrices of the continued pretrained model, we are able to combine the two terms in (13c) by using the basis vectors of U_0 and V_0 as follows:

$$\left\|\hat{\theta}_i'\right\|_{\mathrm{F}}^2 = \left\|\sum_{j=1}^{r_0}\sigma_j^{(0)}u_j^{(0)}v_j^{(0)T} + \sum_{j=1}^{r_0}\sigma_j^{(1')}u_j^{(0)}v_j^{(0)T}\right\|_{\mathrm{F}}^2, \tag{15a}$$

$$= \sum_{j=1}^{r_0}\left[\sigma_j^{(0)} - \sigma_j^{(1')}\right]^2, \tag{15b}$$

where we define an auxiliary symbol

$$\sigma_j^{(1')} = \begin{cases}\sigma_k^{(1)}, & \exists k \in [1, r_1] \text{ s.t. } u_j^{(0)} = u_k^{(1)},\\ 0, & \text{other } k.\end{cases} \tag{16}$$

Both (14) and (15b) correspond to the extreme cases. Since (15b) leads to a smaller denominator, we will use it to examine the error ratio.

Error Ratio. Using (12c) and (15b), a pessimistic error ratio can be approximated and then up bounded as follows:

$$\frac{\|R\|_{\mathrm{F}}^2}{\left\|\hat{\theta}_i'\right\|_{\mathrm{F}}^2} \approx \frac{\sum_{\ell=1}^{r_2}\left[\sigma_\ell^{(2)} + \sigma_\ell^{(1)}\right]^2}{\sum_{j=1}^{r_0}\left[\sigma_j^{(0)} - \sigma_j^{(1')}\right]^2} \tag{17a}$$

$$\leq \frac{r_2 \cdot \max_\ell\left[\sigma_\ell^{(2)} + \sigma_\ell^{(1)}\right]^2}{r_0 \cdot \min_j\left[\sigma_j^{(0)} - \sigma_j^{(1')}\right]^2} \tag{17b}$$

$$= \frac{r_2}{r_0} \cdot \nu, \tag{17c}$$

where $\nu = \max_\ell\left[\sigma_\ell^{(2)} + \sigma_\ell^{(1)}\right]^2 \Big/ \min_j\left[\sigma_j^{(0)} - \sigma_j^{(1')}\right]^2$ is a singular-value based constant. Given that the rank r_2 of the patch is at least one order of magnitude smaller than the rank r_0 of the pretrained model, we conclude the residual matrix term is negligible compared to the naive update term.