PrAd: Prompt Adaptive Tuning for Decoder-only Language Models

Youneng Ma and **Junyi He** and **Haojun Fei**
Qifu Technology
No. 1217, Dong Fang Road, Pudong New Area, Shanghai, China
Building 2, No.6, Jiuxianqiao Road, Chaoyang District, Beijing, China
Correspondence: feihaojun-jk@360shuke.com

Abstract

Fine tuning pretrained language models for downstream NLP tasks, while effective, can be costly when the model size and the number of tasks increase, as it requires full parameter updates and a separate model served for each task. Parameter-efficient tuning (PET) addresses the issue by keeping the pretrained parameters fixed while introducing minimal task-specific parameters. There are two essential PET paradigms: prompt-based tuning and adapter-based tuning, each with distinct limitations. Prompt-based methods suffer from increased input lengths and sensitivity to weight initialization, whereas adapter approaches can substantially increase inference time. To overcome these limitations, we propose prompt adaptive tuning (PrAd), a general prompt-based tuning framework for decode-only models that delivers strong performance with high efficiency, even in multi-task scenarios. Unlike conventional prompt-based tuning which uses soft tokens to "wrap" inputs, PrAd employs adapters for flexible input transformation. While traditional adapter-based tuning adapts both the prompt and decoded tokens, PrAd only adapts the prompt. PrAd enables the creation of diverse prompt-based approaches while providing critical advantages for real-world use: (1) it can maintain original input lengths with easy initialization during training, like adapter-based methods; (2) it can reduce management costs while facilitating deployment and efficient batch inference of different tasks, like prompt-based tuning.; and (3) it introduces no additional inference latency in the decoding phase even when serving multiple tasks concurrently. Experiments on six diverse tasks demonstrate that PrAd can consistently attain comparable or better performance and higher inference efficiency.

1 Introduction

It has been a dominant paradigm to fine tune a pretrained language model (PLM) for the transfer learning of downstream NLP tasks. Though powerful, fine tuning all the parameters and serving a new tuned model for each task can be prohibitively expensive when the number of tasks and model size grow. Parameter-efficient tuning methods (PETs), such as prompt-based tuning (Lester et al., 2021; Li and Liang, 2021), adapter tuning (Houlsby et al., 2019; Pfeiffer et al., 2020; He et al., 2021; Lei et al., 2024; Zhang et al., 2023b), and LoRA (Hu et al., 2021), represent effective approaches to addressing this issue. PETs can attain high parameter sharing by only updating a small number of extra parameters for each task and keeping the pretrained parameters frozen.

Despite their widespread adoption, current mainstream PETs exhibit distinct limitations in practical applications. Prefix Tuning and its variants (Li and Liang, 2021; Wu et al., 2022; Chen et al., 2022; Yang et al., 2023) prepend the input with additional trainable prefixes, which increases the input length and reduces the usable sequence length for downstream tasks. Meanwhile, Prefix Tuning is hard to optimize and its performance changes non-monotonically in trainable parameters (Hu et al., 2021). While adapter-based methods demonstrate superior overall performance and greater stability compared to Prefix Tuning (He et al., 2021; Ding et al., 2023), they are much less efficient in terms of batch inference across diverse tasks, and may incur more management costs and notable additional inference latency in certain scenarios. It has been shown that adapter-based methods can introduce over 20% additional inference latency in an online, short-sequence-length scenario (Hu et al., 2021). Rücklé et al., 2020 propose to drop the adapters in the lower layers of the pretrained models for higher training and inference efficiency. Conditional Adapter (Lei et al., 2024) reduces significant computation by selecting only a small subset of input tokens to query the pretrained model. However, this method is intricate and only applicable

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 4729–4743
November 4-9, 2025

for encoder-only models. LoRA (Hu et al., 2021) introduces no additional inference latency when serving a single task by enabling the merging of its trainable parameters into the pretrained model. However, in multi-task settings, LoRA's efficiency benefit is compromised as task-specific adapters must be maintained separately, preventing weight merging and introducing additional inference latency.

To overcome the limitations of existing PET methods, we introduce PrAd, a novel and efficient PET framework for decoder-only LMs that delivers strong performance while maintaining high efficiency, even in multi-task scenarios. PrAd utilizes adapters exclusively during the refill stage. It provides several key advantages for practical utilization: 1. It can facilitate deployment and batch inference of different tasks, and reduce the management and memory costs incurred by adapters by up to 50% in practical scenarios where the prefill and decoding phases are dis-aggregated (Zhong et al., 2024). 2. Its ease of implementation and training, coupled with robust performance across various NLU and NLG tasks, are pivotal for practical adoption. 3. It does not increase the input length, thereby not reducing the available input length for downstream tasks, addressing a major limitation of existing prompt-based methods. 4. It introduces negligible additional inference latency even when serving multiple tasks concurrently. Our main contributions are summarized as below:

- We propose PrAd, a general prompt-based tuning framework for decode-only models, featuring in "transforming" the prompt with trainable modules only in the prefill phase, giving rise to various novel prompt-based methods. Existing prompt-based approaches can also be conceptualized as specific instances of PrAd.
- We propose sequential and parallel adapter-enhanced prompt tuning strategies which synergistically integrate the advantages of both prompt-based and adapter-based tuning. Our approach is unique in that it does not incur any extra inference latency during decoding while serving multiple tasks concurrently.
- We conduct extensive experiments across various tasks to validate the effectiveness of our methods. PrAd stands out for causing negligible extra inference latency in single-task settings and far greater efficiency in multi-task scenarios while still delivering competitive performance and facilitating deployment. Our code is publicly available [1].

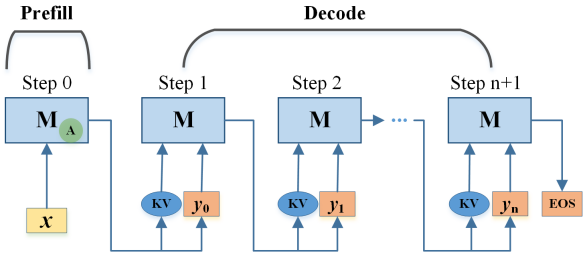


Figure 1: PrAd Inference. "**M**" represents the pretrained LM. "**A**" denotes inserted adapters. The prompt and output tokens are represented by $\boldsymbol{x}$ and $\boldsymbol{y_i}$ respectively. "**KV**" denotes the key-value caches. "**EOS**" is the stop token. PrAd differs from traditional adapter-based approaches by utilizing adapters exclusively during the prefill stage. It slashes the management and memory overhead associated with adapters by **50%** in decoupled deployments, while introducing **zero** additional inference latency during decoding.

2 Background

2.1 Decoder-only language model

The Transformer model (Vaswani et al., 2017) is now the most widely-used architecture for the majority of successful PLMs. A standard Transformer model consists of an encoder and a decoder. Decoder-only LMs only use the decoder for pretraining with the task of language modeling (Radford et al., 2018). A typical Decoder-only LM comprises of an Embedding layer, followed by a stack of Transformer Blocks and a LM head at the end. Each Transformer Block comprises of two sub-layers: Multi-head Self-Attention Layer (ATTN) and a fully connected Feed-Forward Network (FFN). Decoder-only LMs demonstrated strong performance on many NLP tasks in the zero-shot and few shot settings (Brown et al., 2020) and have garnered popularity nowadays. Formulating various NLP tasks as conditional Natural Language Generation (NLG) problems has been a popular practice in NLP as it allows us to use a unified learning framework while maintaining competitive performance across various tasks (Raffel et al., 2020). Let us denote $\boldsymbol{x} = [x_0, x_1, ..., x_m]$ as the input (or prompt), and $\boldsymbol{y} = [y_0, y_1, ..., y_n]$ as the output sequence. A decoder-only LM such as (Radford et al., 2019; Brown et al., 2020; Touvron et al., 2023) generates the target sequence

[1] Code: https://github.com/younengma/prad

auto-regressively as below:

$$y_i = \mathbf{LM}([\boldsymbol{x}, \boldsymbol{y}_{<i}]) \quad (1)$$

The inference process comprises of two phases: **Prefill**: encode the input and generate the first output token. **Decode**: generate the rest of tokens based on tokens that come before them. During the inference process, as each token only depends on past tokens, the keys and values of the past tokens can be cached and reused for the generation of future tokens. The computation of prefill phase and decode phase differs in latency preference for different forms of parallelism. It is often beneficial to dis-aggregate those two phases for optimized service serving in real scenarios (Zhong et al., 2024).

2.2 Parameter-efficient Tuning

Nowadays, as pretrained models become increasingly larger, fine tuning all parameters and serving a tuned model for each task can be prohibitively expensive, especially when a large number of tasks are involved. PET is an effective approach to mitigate the issue. The main idea of PET is to fix the pretrained model's parameters and introduce a small set of extra trainable parameters for task adaptation, yielding high parameters sharing. PETs are promising for several reasons. Firstly, they can often perform comparable with Fine Tuning while being much more parameter efficient (Houlsby et al., 2019; Li and Liang, 2021; Hu et al., 2021; He et al., 2021). Secondly, many of these methods support efficient batch inference across multiple tasks conveniently, a particularly valuable feature for large-scale applications like cloud services where simultaneous processing of numerous tasks is essential. Finally, in low-resource settings, PETs demonstrate superior extrapolation performance than Fine Tuning (Li and Liang, 2021). Many innovative PETs have been introduced and summarized in the literature (Ding et al., 2023; Han et al., 2024). Here, we introduce the methods that are particularly pertinent to our work.

Prompt-based tuning: A prompt usually refers to the input to a LM at the input layer. Broadly speaking, a prompt can also refer to an input to deep layers of the LM. Prompt-based tuning features in wrapping the input with additional tokens to convert various downstream NLP tasks to a unified language modeling task (Brown et al., 2020; Liu et al., 2023). It is a powerful and attractive paradigm as it allows the LM to perform few-shot or even zero-shot learning with plain texts. However, designing manual prompts or searching for discrete prompts can be time-consuming and sub-optimal (Liu et al., 2022a). Some works propose to combine soft trainable prompts with text prompts (Lester et al., 2021; Li and Liang, 2021; Liu et al., 2022b). Prompt Tuning (Lester et al., 2021) inserts soft prompts in the input layer and it become more competitive as the model size grows. Instead of adding prompts only at the input layer, Prefix Tuning (Li and Liang, 2021) introduces trainable continuous tokens (prefixes) to at every Transformer layer, and significantly enhances model performance. Instead of using fixed prefixes, recent methods propose to incorporate dynamic prefixes that are dependent on input instances (Chen et al., 2022; Wu et al., 2022; Yang et al., 2023). These methods have shown enhanced performance, albeit with an increase in complexity. An advantage of prompt-based methods over other methods is that they usually do not modify the architecture of the PLM, a feature that streamlines the implementation and makes deployment in real-world scenarios more convenient. Although prompt-based methods have shown very competitive performance in various downstream tasks, they are relatively difficult to optimize especially for tasks with long input and generally converge slower (Li and Liang, 2021; Hu et al., 2021; Chen et al., 2022). Meanwhile, adding prefixes to the input increases the input length and reduces the available sequence length for downstream tasks.

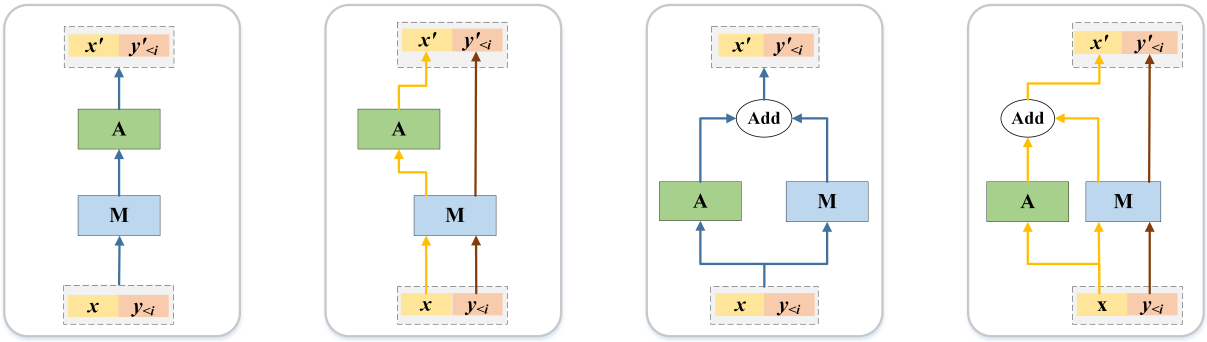
Adapter-based tuning: Adapter-based tuning approaches (Rebuffi et al., 2017; Zhu et al., 2021; He et al., 2021) insert lightweight neural modules, termed adapters, into the pre-trained model and tune only the adapters for task adaptation. From a broader perspective, all PETs can be conceptually seen as specialized forms of adapter-based approaches, as they fundamentally share the principle of incorporating lightweight neural modules for task adaptation, despite variations in module design and application. Adapters were first proposed by Rebuffi et al., 2017 for domain transfer learning in computer vision. Houlsby et al., 2019 then applied adapters for efficient NLP transfer learning and designed a popular adapter architecture which comprises of a down projection W_d and a up projection W_u with a residual connection for the Transformer Layer (Vaswani et al., 2017) as below:

$$\mathbf{A}(\boldsymbol{x}) = ReLU(\boldsymbol{x}W_d)W_u + \boldsymbol{x} \quad (2)$$

Adapter-based methods are very competitive as they can often perform on par (or only slightly under-perform) compared with Fine Tuning (Houlsby et al., 2019; Lin et al., 2020; Rücklé et al., 2020; He et al., 2021). There are two types of adapters: Sequential Adapter (SA) (Houlsby et al., 2019) and Parallel Adapter (PA) (He et al., 2021; Zhu et al., 2021). SA is inserted after the PLM module, the PLM module and the adapter process the input sequentially. In contrast, PA is inserted beside the PLM module, the PLM module and the adapter process input in parallel. An graphical illustration of SA and PA is depicted in Figure 2(a) and (c). PA has demonstrated better performance than SA in many scenarios (He et al., 2021; Zhu et al., 2021). In terms of computational efficiency, compared with standard Fine Tuning, adapters can be up to 60% faster in training while being 4-6% slower at inference on average (Rücklé et al., 2020). However, the inference latency introduced by adapters can be significant (>20%) in an online, short-sequence-length scenario (Hu et al., 2021). Rücklé et al., 2020 showed that dropping the adapters from lower transformer layers can improve the inference speed considerably in multi-task settings. Conditional Adapter (Lei et al., 2024) attained an impressive 2x to 8x inference by selecting only a small subset of input tokens to be processed by the slower pretrained model and all tokens processed by the fast adapter layer. However, this method is only applicable to encoder models and introduces significant complexity. Inspired by that fact that pretrained models reside on a low intrinsic dimension (Li et al., 2018; Aghajanyan et al., 2020), Hu et al., 2021 proposed Low-Rank Adaptation (LoRA) approach which injects trainable decomposition low-rank matrices into each layer of the Transformer architecture for task adaptation. LoRA can be regarded as a special case of PA. One key advantage of LoRA is that task-specific weights can be merged into the pretrained model, introducing no additional inference latency. However, this benefit is lost in multi-task scenarios, where different tasks require separate LoRA weights and merging is no longer feasible. Similarly, in single-task settings involving concurrent deployment of multiple adapter versions—such as during A/B testing—weight merging cannot be applied, limiting LoRA's deployment flexibility. LoRA has gained much popularity recently and has been further developed (Valipour et al., 2022; Zhang et al., 2023a; Liu et al., 2024).

3 Our Method

We introduce Prompt Adaptive Tuning (PrAd) for decoder-only LMs, a novel and versatile prompt-based framework which extends existing prompt-based approaches while integrating the advantages of adapter-based methods. The core innovation of PrAd lies in its general and dynamic input transformation mechanism during the prefill phase, formally expressed as: $\boldsymbol{x'} = \mathbf{A}(\boldsymbol{x})$, where "$\mathbf{A}$" represents a parameterized prompt transformation module, $\boldsymbol{x}$ denotes input representation. As illustrated in Figure 1, the inference of decoder-only LMs comprises of two distinct operational phases: prefill and decode. Disaggregating these two phases can often lead to optimized service performance and efficiency (Zhong et al., 2024). We decouple these phases into separate computational models - specifically, a prefill model and a decode model. PrAd exclusively modifies the prefill model by incorporating "$\mathbf{A}$" into its structure for task adaptation, while keeping the decode model intact. Notably, conventional prompt-based methods (Li and Liang, 2021; Chen et al., 2022; Wu et al., 2022; Yang et al., 2023) can also be regarded as special instances of PrAd. PrAd offers substantial versatility in the choice of "$\mathbf{A}$", enabling the derivation of numerous PrAd variants. However, selecting an appropriate "$\mathbf{A}$" remains crucial for achieving successful outcomes. Empirical findings from existing prompt-based approaches indicate that an effective transformation "$\mathbf{A}$" should create a strong dependency between $\boldsymbol{x}$ and $\boldsymbol{x'}$. Human-designed prompts should be relevant to the input topic (Brown et al., 2020). The soft prefixes should be initialized using task-specific token embeddings for more stable training (Li and Liang, 2021). Incorporating input-dependent prefixes has been shown to improve model performance (Chen et al., 2022; Wu et al., 2022; Yang et al., 2023). Prompt-based methods are closely correlated with adapter-based methods. He et al., 2021 demonstrated that the underlying mechanism of prompt-based tuning can be can be mathematically formulated as implementing PA in the ATTN layer. However, unlike adapter-based methods that can employ identity initialization in the residual adapters for stable training (Houlsby et al., 2019), Prefix Tuning lacks the residual structure, which may partially account for the optimization difficulties observed in Prefix Tuning. Conventional prompt-based methods primarily perform input transformations limited to the ATTN layer via



(a) Sequential Adapter (b) Sequential PrAd (c) Parallel Adapter (d) Parallel PrAd

Figure 2: Graphical illustration of the computation of standard adapters and the proposed PrAd variants when decoding the i^{th} target token. Here, "**M**" represents a certain sublayer of the PLM that is frozen. "**A**" denotes an inserted trainable adapter that works together with "**M**". "$\boldsymbol{x}$" denotes the input hidden states related to prompt. "$\boldsymbol{y_{<i}}$" represents the input hidden states related to the already decoded tokens. "$\boldsymbol{x'}$" and "$\boldsymbol{y'_{<i}}$" represent the output hidden states. Standard adapters process both "$\boldsymbol{x}$" and "$\boldsymbol{y_{<i}}$" with "**M**" and "**A**" as denoted in (a) and (c). In contrast, the proposed methods process "$\boldsymbol{x}$" with "**M**" and "**A**", but process "$\boldsymbol{y_{<i}}$" only with "**M**" as shown in (b) and (d).

'additional attention values'. As Wang et al., 2024 theoretically proves, the limitation makes them less adaptable than methods employing flexible direct input transformations. This rigidity may explain why prompt-based methods often underperform adapter methods on challenging tasks.

Building upon these theoretical insights and empirical observations, we propose the adapter-based prompt transformation, leveraging residual adapters to enable powerful input transformation, while enforcing a strong dependence relationship between $\boldsymbol{x}$ and $\boldsymbol{x'}$ and enabling identity initialization for stable training. When decoding the i^{th} token of target sequence, the input is the concatenation of prompt $\boldsymbol{x}$ and the already decoded token sequence $\boldsymbol{y_{<i}}$. Let us denote $\boldsymbol{x'}$, $\boldsymbol{y'_{<i}}$ as the corresponding outputs of $\boldsymbol{x}$, $\boldsymbol{y_{<i}}$ respectively. A standard parallel adapter (He et al., 2021) computes its output together with a pretrained module as below:

$$[\boldsymbol{x'}, \boldsymbol{y'_{<i}}] = \mathbf{M}([\boldsymbol{x}, \boldsymbol{y_{<i}}]) + \mathbf{A}([\boldsymbol{x}, \boldsymbol{y_{<i}}]) \quad (3)$$

where "**M**" denotes the frozen pretrained module and "**A**" denotes the adapter. As we can see in the function, the standard parallel adapter processes both the prompt $\boldsymbol{x}$ and the already decoded tokens $\boldsymbol{y_{<i}}$. Inspired by the fact a PLM can be effectively steered only by prompts, we propose to apply a parallel adapter only on the prompt $\boldsymbol{x}$:

$$[\boldsymbol{x'}, \boldsymbol{y'_{<i}}] = \mathbf{M}([\boldsymbol{x}, \boldsymbol{y_{<i}}]) + [\mathbf{A}(\boldsymbol{x}), 0] \quad (4)$$

As the proposed approach incorporates a parallel adapter that exclusively processes the prompt, it's termed Prompt Adaptive Tuning with Parallel Adapter (PrAd-PA). Accordingly, we also propose the sequential variant of PrAd (PrAd-SA). A conventional sequential adapter (Houlsby et al., 2019), together with a pretrained module, processes the input as below:

$$[\boldsymbol{x'}, \boldsymbol{y'_{<i}}] = \mathbf{A}(\mathbf{M}([\boldsymbol{x}, \boldsymbol{y_{<i}}])) \quad (5)$$

Applying the sequential adapter only on the prompt $\boldsymbol{x}$ yields PrAd-SA:

$$[\boldsymbol{x'}, \boldsymbol{y'_{<i}}] = [\mathbf{A}(\mathbf{M}(\boldsymbol{x}, \boldsymbol{y_{<i}})[:l]), \mathbf{M}(\boldsymbol{x}, \boldsymbol{y_{<i}})[l:]] \quad (6)$$

where l is the length of $\boldsymbol{x}$, tensor slicing operations (eg."$[:\ l]$" and "$[l\ :]$") follow the rule of PyTorch. Figure 2 depicts the different computation processes between standard adapters and the proposed methods. The key distinction between PrAd and prior adapter methods (Houlsby et al., 2019; He et al., 2021) is that adapters of PrAd only process the prompt, while adapters of previous methods process both the prompt and already decoded tokens. Our design eliminates adapter requirements during decoding, reducing their memory footprint and management overhead by 50% in prefill-decode deployments while also boosting decoding efficiency. PrAd can also be viewed as a dynamic prompt tuning method (Wu et al., 2022), as its adapters generate instance-specific prompts for modules after them. However, unlike previous methods, PrAd maintains the original input length and offers broader applicability as it's not restricted to ATTN layer. PrAd introduces no additional inference latency during decoding even in multi-task settings. As far as we know, this is only PET method

that possesses this merit. Although this article primarily focuses on adapter-based methods given their effectiveness and broad usage, it is worth noting that other PrAd variants can be easily derived. In most cases, for an existing PET method, we can apply it exclusively to the prefill model to develop a corresponding PrAd variant. For instance, by only tuning the bias terms of the prefill model using Bitfit (Zaken et al., 2021), we can derive PrAd-Bitfit. Similarly, applying DoRA (Liu et al., 2024) only during the prefill phase yields PrAd-DoRA.

4 Experiments

4.1 Tasks, datasets and metrics

To validate the effectiveness of the proposed methods, we conduct experiments on both NLU and NLG tasks with six diverse datasets as below:

SST2: A sentence-level binary classification task. We use the binary version Stanford Sentiment Treebank (Socher et al., 2013), a corpus of movie reviews with human annotations of their sentiment. The model needs to predict the sentiment label (positive/negative) of a given sentence. We report the accuracy metric on the development set.

MNLI: A Natural Language Inference NLU task. We use the Multi-Genre Natural Language Inference Corpus (Williams et al., 2017). The model needs to determine whether a premise-hypothesis pair is entailment, neutral or contraction. We report the accuracy score of the matched section on the development set.

E2E: A table-to-text NLG task. We use the E2E datasets (Novikova et al., 2017) which contains around 50K examples with 8 distinct fields in the restaurant domain. The model needs to generate natural language responses based on attributes. We report BLEU (Papineni et al., 2002), NIST (Belz and Reiter, 2006), METEOR (MET) (Banerjee and Lavie, 2005), ROUGE-L (Lin, 2004) and CIDEr (Vedantam et al., 2015) with the official evaluation script.

WebNLG: A table-to-text NLG task. We use the WebNLG-challenge (Gardent et al., 2017) dataset which contains triplets-text pairs of 9 categories in the train and validation set and five extra unseen categories in the test set. The model needs to generate sensible sentences based on short triplets. We report BLEU, MET and TER (Snover et al., 2005) for All (A), Seen (S) and Unseen (U) categories on the test set following notations in Prefix Tuning (Li and Liang, 2021).

MT: A German to English translation task. We use the IWSLT2014 GE-EN dataset. We report BLEU.

XSUM: An English summarization task. We use XSum (Narayan et al., 2018) dataset. The model needs to predict a summary given a news article. We report ROUGE-1/2/L (Lin, 2004).

4.2 Baselines and proposed model variants

To validate the proposed methods, we use the following representative methods as the baselines:

Fine Tuning: Fine Tuning is a classic approach for adaptation which updates all the parameters in the pretrained model. It can show a relative upper bound performance of the tasks.

Prefix Tuning: Prefix Tuning (Li and Liang, 2021) is a seminar work of prompt-based tuning methods. It prepends trainable prefix vectors to the keys and values of every ATTN layer for task adaptation. We use the re-parametrization trick (Li and Liang, 2021) in all the experiments for stable training.

PA: Parallel Adapter inserts an adapter in parallel into an existing sublayer. We focus on the application of parallel adapters in the FFN layer as it attains better performance (He et al., 2021).

LoRA: LoRA is a special kind of PA where trainable pairs of rank decomposition matrices are inserted in parallel to existing weight matrices. By default, we mainly focus on the basic variant which applies LoRA to query, value projection matrices in the ATTN layer (Hu et al., 2021).

SA: Sequential Adapter inserts an adapter after an existing sublayer (Houlsby et al., 2019). We focus on the application of sequential adapter in the FFN layer for simplicity.

We mainly consider the following variants of the proposed PrAd methods and provide experiment results of additional PrAd variants in Appendix C.2.

PrAd-PA: The "PA version" of PrAd. Like PA, we insert an adapter into the FFN layer. However, in PA, the inserted adapter processes both the prompt $\boldsymbol{x}$ and the decoded tokens $\boldsymbol{y}_{<i}$ as shown in Figure 2(c). In PrAd-PA, the inserted adapter processes only $\boldsymbol{x}$ as shown in Figure 2(d).

PrAd-LoRA: In LoRA, the inserted adapter processes both $\boldsymbol{x}$ and $\boldsymbol{y}_{<i}$ as shown in Figure 2(c). In PrAd-LoRA, the inserted adapter only processes $\boldsymbol{x}$ as shown in Figure 2(d). By default, we apply LoRA only in the ATTN layer.

PrAd-SA: The sequential version of PrAd. While adapters in SA process both $\boldsymbol{x}$ and $\boldsymbol{y}_{<i}$ as depicted in Figure 2(a), adapters in PrAd-SA only process $\boldsymbol{x}$ as shown in Figure 2(b).

	SST2	MNLI	E2E					WebNLG		
	Accuracy	Accuracy	BLEU	NIST	MET	ROUGE-L	CIDEr	BLEU	MET	TER↓
Fine Tuning	91.36$_{.73}$	81.52$_{.18}$	68.22$_{.70}$	8.742$_{.058}$	0.464$_{.001}$	70.78$_{.53}$	2.441$_{.014}$	45.33$_{.88}$	0.375$_{.002}$	0.534$_{.013}$
Prefix Tuning	91.32$_{.42}$	68.31$_{.81}$	68.53$_{.06}$	8.743$_{.009}$	**0.465**$_{.001}$	**71.41**$_{.13}$	**2.462**$_{.005}$	**52.95**$_{.41}$	0.405$_{.001}$	**0.442**$_{.004}$
PA	**91.74**$_{.34}$	80.00$_{.36}$	67.76$_{.40}$	8.637$_{.054}$	0.464$_{.001}$	70.27$_{.17}$	2.382$_{.031}$	50.69$_{.33}$	0.402$_{.001}$	0.465$_{.003}$
LoRA	90.83$_{.73}$	80.32$_{.25}$	68.35$_{.27}$	**8.761**$_{.010}$	0.464$_{.002}$	70.62$_{.28}$	2.427$_{.024}$	52.07$_{.36}$	**0.406**$_{.001}$	0.448$_{.006}$
PrAd-PA	91.63$_{.09}$	79.96$_{.23}$	68.40$_{.03}$	8.736$_{.028}$	0.462$_{.002}$	70.91$_{.02}$	2.396$_{.004}$	51.38$_{.21}$	0.400$_{.001}$	0.454$_{.006}$
PrAd-LoRA	90.44$_{.42}$	**80.74**$_{.24}$	**68.55**$_{.20}$	8.744$_{.025}$	0.464$_{.001}$	70.95$_{.23}$	2.397$_{.007}$	51.72$_{.64}$	0.398$_{.001}$	0.465$_{.009}$

Table 1: Performance on tasks SST2, MNLI, E2E and WebNLG with GPT-2 Small. Here, we report metrics for "All categories" on WebNLG. The PETs have similar sizes of parameters to store during inference. We run each experiment 3 times with random seeds and report the mean value with standard deviation in the subscript. The best scores among PETs are **boldfaced** and second best scores are underlined.

	GPT-2 Large				LLaMA-7B			
	SST2	MNLI	MT	XSUM	SST2	MNLI	MT	XSUM
	Accuracy	Accuracy	BLEU	ROUGE-1/2/L	Accuracy	Accuracy	BLEU	ROUGE-1/2/L
PA	94.53$_{.14}$	85.64$_{.34}$	33.94$_{.23}$	**38.32**$_{.03}$/**16.70**$_{.01}$/**31.14**$_{.01}$	96.90$_{.16}$	90.14$_{.03}$	**40.73**$_{.17}$	**43.94**$_{.11}$/**21.59**$_{.12}$/**36.15**$_{.14}$
LoRA	94.42$_{.11}$	85.38$_{.35}$	33.13$_{.06}$	37.89$_{.13}$/16.36$_{.08}$/30.77$_{.09}$	96.90$_{.25}$	90.45$_{.08}$	40.42$_{.16}$	43.66$_{.12}$/21.32$_{.15}$/35.90$_{.15}$
PrAd-PA	**94.65**$_{.14}$	**85.73**$_{.18}$	**34.35**$_{.13}$	38.13$_{.13}$/16.49$_{.11}$/31.01$_{.13}$	**97.05**$_{.11}$	90.30$_{.04}$	40.56$_{.16}$	43.88$_{.10}$/21.50$_{.08}$/**36.15**$_{.09}$
PrAd-LoRA	94.27$_{.34}$	85.35$_{.07}$	33.00$_{.09}$	37.92$_{.16}$/16.28$_{.10}$/30.80$_{.10}$	96.90$_{.25}$	**90.46**$_{.14}$	40.46$_{.19}$	43.78$_{.11}$/21.40$_{.20}$/36.05$_{.16}$

Table 2: Performance on tasks SST2, MNLI, MT and XSUM with GPT-2 Large and LLaMA-7B.

	MT	XSUM
	BLEU	ROUGE-1/2/L
Fine Tuning	31.18$_{.04}$	35.11$_{.10}$/13.97$_{.09}$/28.17$_{.07}$
Prefix Tuning	16.72$_{.16}$	30.84$_{.08}$/10.49$_{.05}$/24.41$_{.06}$
SA	29.92$_{0.1}$	33.38$_{.09}$/12.67$_{.05}$/26.77$_{.07}$
PA	30.63$_{.16}$	34.09$_{.09}$/**13.24**$_{.04}$/27.34$_{.05}$
LoRA	29.57$_{.11}$	33.72$_{.04}$/12.70$_{.03}$/26.94$_{.00}$
LoRA-FFN	**31.18**$_{.03}$	33.93$_{.04}$/13.01$_{.02}$/27.21$_{.07}$
PrAd-SA	27.71$_{.14}$	33.71$_{.07}$/12.83$_{.04}$/27.09$_{.07}$
PrAd-PA	29.40$_{.19}$	**34.19**$_{.06}$/13.20$_{.06}$/**27.55**$_{.07}$
PrAd-LoRA	26.67$_{.33}$	33.61$_{.02}$/12.72$_{.04}$/27.00$_{.03}$
PrAd-LoRA-FFN	29.96$_{.09}$	34.11$_{.06}$/13.19$_{.03}$/27.49$_{.05}$

Table 3: Performance on tasks MT and XSUM with GPT-2 Small.

4.3 Experiment settings

We use GPT-2 Small (124M), GPT-2 Large (774M) (Radford et al., 2019) and LLaMA-7B (Touvron et al., 2023) as the PLMs. We formulate all tasks as NLG problems. For example, instead of adding a classification head for label index prediction, we directly generate the sentiment label (positive/negative) conditioned on the input text for SST2. We mainly focus on the variants of parallel PrAds and their counterparts, PA and LoRA given their superiority (He et al., 2021; Zhu et al., 2021; Ding et al., 2023). As a general rule, we use a smaller parameter budget for easier tasks (e.g. SST2 and MNLI) and larger parameter budget for harder tasks (e.g. MT and XSUM). For a fair comparison, the sizes of the parameters to store during inference of different PETs are set to be nearly the same for the same task. For data processing, we follow the sample practice by Lin et al., 2020, which inserts task tokens into the input to help the model better understand the semantic structure of the input. We run the experiments with 3 random seeds and report the average score with standard deviation in the subscript. More experiment details are provided in the Appendices A and B.

4.4 Results and Analysis

GPT-2 small: The performance on tasks SST2, MNLI, E2E and WebNLG is presented in Table 1. For NLU tasks SST2 and MNLI, the proposed PrAd methods consistently achieve comparable or superior performance relative to PA and LoRA, thereby validating their effectiveness. In contrast, Prefix Tuning exhibits a substantial performance degradation on MNLI, suggesting inherent optimization challenges of the method. For E2E, Prefix Tuning achieves the best overall performance on the task. The performance gap of Prefix Tuning on MNLI and E2E demonstrates that a method which underperforms on one task may excel on another, underscoring the importance of evaluating methods across diverse tasks. Our methods keep performance parity. Notably, PrAd-LoRA achieves the highest BLEU score and exhibits superior performance compared to both PA and LoRA, suggesting that our approaches can outperform conventional adapter-based methods in specific task. For WebNLG, PETs generally outperform Fine Tuning and the proposed methods perform comparable with baselines. Table 3 presents the results on the more challenging XSUM and MT tasks. Prefix Tuning exhibits a significant drop in performance.

Number of Tasks	8				16				64			
Inference Phase	Prefill	Decode	Total	Speedup	Prefill	Decode	Total	Speedup	Prefill	Decode	Total	Speedup
LoRA	27.54	1747.91	1775.05	-	45.05	2774.53	2819.59	-	140.39	8588.22	8728.61	-
PA	15.67	937.09	952.76	×0.86	22.69	1329.27	1351.96	×1.09	70.13	3429.28	3499.41	×1.49
PrAd-LoRA	27.54	482.05	509.59	×2.48	45.05	496.56	541.62	×4.21	140.39	513.29	653.68	×12.35
PrAd-PA	15.67	482.05	**497.72**	×2.57	22.69	496.56	**519.25**	×4.43	70.13	513.29	**583.42**	×13.96

Table 4: Inference latency with GPT-2 Large measured in milliseconds averaged across 50 runs on an Nvidia A100 40G in multi-task scenarios. In each run, we simultaneously encode multiple task samples of 64-token length and generate output sequences of equal length. The number of trainable parameters of different PETs are set nearly the same and the bottleneck dimension in the adapters is 16. The speedup is compared with LoRA. The shortest inference time among the PETs is shown in **bold**, while the second shortest is underlined.

While our methods show slightly inferior performance to PA and LoRA on MT, they demonstrate improved performance on XSUM. Notably, PrAd-PA outperforms all baselines on XSUM. Moreover, the results reveal that implementing LoRA in the FFN layer (LoRA-FFN and PrAd-LoRA-FFN) yields better performance, which aligns with previous research (He et al., 2021).

GPT-2 Large and LLaMA-7B: We further conduct experiments on 4 tasks: SST2, MNLI, XSUM and MT with GPT-2 Large and LLaMA-7B. We only compare our methods with LoRA and PA given their superiority (He et al., 2021; Hu et al., 2021; Ding et al., 2023). Table 2 presents the performance on the 4 tasks. Our method matches or exceeds strong baselines (PA/LoRA). On GPT-2 Large, PrAd-PA emerges as the top-performing method across three tasks: SST-2, MNLI, and MT, while securing the second-best performance in XSUM. PrAd-LoRA maintains comparable performance against the baselines. On LLaMA-7B, PrAd-LoRA achieves optimal performance in MNLI, while PrAd-PA ranks first in SST2 and ranks second in both MT and XSUM tasks. Notably, while our methods underperform on MT with GPT-2 Small (as shown in Table 3), they can outperform the baselines on the large models, suggesting that our approaches become more competitive with the increase in model scale.

Inference efficiency: Prior research (Hu et al., 2021; Rücklé et al., 2020) has found that adapters can introduce significant additional inference latency in certain scenarios, which is also observed in our experiments. Figure 3 presents the inference time of different methods with various sequence lengths and batch sizes with GPT-2 Small in single task settings. PrAd-LoRA, like LoRA, introduces no extra inference latency, whereas PrAd-PA and Prefix Tuning have minimal impact. PA, however, leads to a significant latency increase. These findings are further corroborated by our extended experiments on large models in Appendix C.1. Table 4 presents the inference time when handling multiple tasks simultaneously. The results show our methods achieves substantially higher efficiency than PA and LoRA. While LoRA achieves high inference efficiency in single-task settings, it is much less efficient than our approach in multi-task scenarios due to the use of task-specific adapters during both prefill and decoding phase, which prevents effective batched computations. In contrast, as PrAd doesn't use any task-specific modules in the decoding phase, it attains high inference efficiency in multi-task environments by enabling batched inference across tasks and introducing almost no additional computational overhead during decoding.

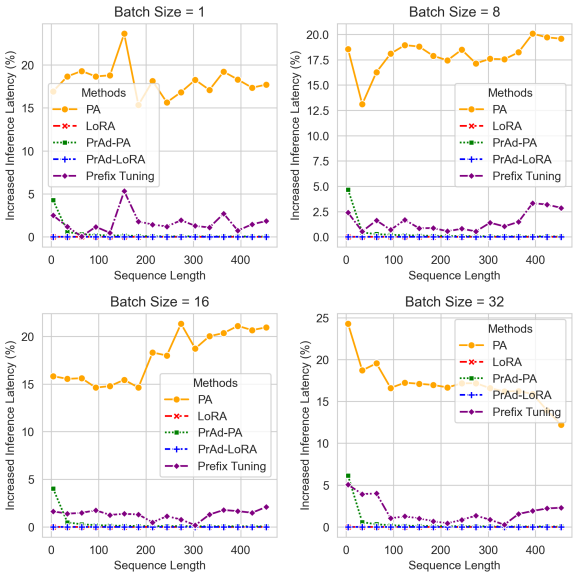


Figure 3: The increased inference latency (%) compared with Fine Tuning averaged over 50 runs on an NVIDIA A100 40G with GPT-2 Small. In each run, we encode an input of "Sequence Length" and generate an output sequence of the same length. The prefix length and bottleneck dimension in the adapters are set as 32.

Management costs reduction: In real-world de-

ployments, the prefill and decode phases are often decoupled for service optimization. PrAd introduces adapters exclusively during the prefill phase, leaving the decode model unchanged throughout the adapter lifecycle. This design confines adapter maintenance—including updates, removals, and version control—to the prefill model only, resulting in a 50% reduction in adapter management overhead.

5 Conclusion

We propose PrAd, a general prompt-based tuning framework for decoder-only LMs. PrAd enables the development of diverse prompt-based methods and integrates the strengths of both prompt-based and adapter-based tuning. PrAd (1) maintains input length and allows simple initialization during training, while performing on par with or surpassing strong baselines (e.g., PA, LoRA) across diverse tasks; 2) introduces zero additional inference latency during decoding, even when serving multiple tasks concurrently; 3) facilitates deployment and batch inference of various tasks, reducing 50% management and memory costs of adapters while significantly boosting inference efficiency in multi-task scenarios. With its strong performance, greater efficiency and lower management costs, PrAd can be a superior choice for real-world use, especially in multi-task scenarios.

6 Limitations

As our methods use additional adapters in the prefill phase during inference, it can introduce additional computation cost when generating the first token. While the overall performance of our method remains competitive with other strong PET techniques in most scenarios, it may underperform compared to state-of-the-art methods depending on task-specific characteristics. As studied in previous works, the choice of hyper-parameters in adapter-based methods can have an impact on the final performance (Valipour et al., 2022; Zhang et al., 2023a), further research is required to understand the impact of the choice of hyper-parameters in our methods.

Additionally, our work currently focuses only on decoder-only models, although the concept of prompt adaptive tuning could potentially be extended to other architectures involving autoregressive decoding (e.g., encoder-decoder models), its efficacy across different architectures remains an open question that we leave for future work.

References

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*.

Satanjeev Banerjee and Alon Lavie. 2005. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72.

Anja Belz and Ehud Reiter. 2006. Comparing automatic and human evaluation of nlg systems. In *11th conference of the european chapter of the association for computational linguistics*, pages 313–320.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Yifan Chen, Devamanyu Hazarika, Mahdi Namazifar, Yang Liu, Di Jin, and Dilek Hakkani-Tur. 2022. Inducer-tuning: Connecting prefix-tuning and adapter-tuning. *arXiv preprint arXiv:2210.14469*.

Ning Ding, Yujia Qin, Guang Yang, Fu Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Haitao Zheng, Jianfei Chen, Y. Liu, Jie Tang, Juanzi Li, and Maosong Sun. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5:220–235.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez Beltrachini. 2017. The webnlg challenge: Generating text from rdf data. In *Proceedings of the 10th international conference on natural language generation*, pages 124–133.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *ArXiv*, abs/2403.14608.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

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, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Tao Lei, Junwen Bai, Siddhartha Brahma, Joshua Ainslie, Kenton Lee, Yanqi Zhou, Nan Du, Vincent Zhao, Yuexin Wu, Bo Li, et al. 2024. Conditional adapters: Parameter-efficient transfer learning with fast inference. *Advances in Neural Information Processing Systems*, 36.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. 2018. Measuring the intrinsic dimension of objective landscapes. *arXiv preprint arXiv:1804.08838*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Zhaojiang Lin, Andrea Madotto, and Pascale Fung. 2020. Exploring versatile generative language model via parameter-efficient transfer learning. *arXiv preprint arXiv:2004.03829*.

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

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

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. *arXiv preprint arXiv:2402.09353*.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022b. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Annual Meeting of the Association for Computational Linguistics*.

Shashi Narayan, Shay B Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *arXiv preprint arXiv:1808.08745*.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The e2e dataset: New challenges for end-to-end generation. *arXiv preprint arXiv:1706.09254*.

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.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 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.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. *Advances in neural information processing systems*, 30.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2020. Adapterdrop: On the efficiency of adapters in transformers. *arXiv preprint arXiv:2010.11918*.

Mathew Snover, Bonnie Dorr, Richard Schwartz, John Makhoul, Linnea Micciulla, and Ralph Weischedel. 2005. A study of translation error rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas (AMTA 06)*, pages 223–231.

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 empirical methods in natural language processing*, pages 1631–1642.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. 2022. Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. *arXiv preprint arXiv:2210.07558*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. 2015. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4566–4575.

Yihan Wang, Jatin Chauhan, Wei Wang, and Cho-Jui Hsieh. 2024. Universality and limitations of prompt tuning. *Advances in Neural Information Processing Systems*, 36.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Zhuofeng Wu, Sinong Wang, Jiatao Gu, Rui Hou, Yuxiao Dong, V. G. Vinod Vydiswaran, and Hao Ma. 2022. Idpg: An instance-dependent prompt generation method. In *North American Chapter of the Association for Computational Linguistics*.

Xianjun Yang, Wei Cheng, Xujiang Zhao, Wenchao Yu, Linda Petzold, and Haifeng Chen. 2023. Dynamic prompting: A unified framework for prompt tuning. *arXiv preprint arXiv:2303.02909*.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023a. Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*. Openreview.

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

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. *arXiv preprint arXiv:2401.09670*.

Yaoming Zhu, Jiangtao Feng, Chengqi Zhao, Mingxuan Wang, and Lei Li. 2021. Serial or parallel? plug-able adapter for multilingual machine translation. *arXiv preprint arXiv:2104.08154*, 6(3).

A Experiment Details for GPT-2 Models

We use "huggingface transformers" and PyTorch packages for the implementation of different methods. For all tasks we use linear scheduler with warm-up ratio of 0.06 and grad norm is set as 1. We use beam search for decoding during inference, with beam size set as 1 for MNLI and SST2, beam size as 3 for the rest of the tasks. The training hyperparameters are selected as follows: to maintain consistency, we standardize the batch size and epoch settings across all methodologies for a given task. The learning rate for Fine Tuning is set at the widely-adopted value of 5e-5. In the case of PETs, the learning rate is carefully chosen within the spectrum of 1e-4 to 3e-3, guided by the validation loss metrics and settings in previous works (Lin et al., 2020; Hu et al., 2021; He et al., 2021).

GPT-2 Small: The hyper parameters used for GPT-2 Small are shown in Table 5. For The bottleneck dimension of adapters is set as 16 in SST2 and 32 in MNLI, which results in around 0.24% trainable parameters in SST and 0.48% in MNLI for the PETs. For E2E challenge and WebNLG, a bottleneck dimension of 108 is used for the adapters, around 1.6% trainable parameters are introduced in PETs. For XSUM and MT tasks, a bottleneck dimension of 200 used for the adapters, around 2.9% trainable parameters are introduced. The prefix length in Prefix Tuning and the rank of introduced matrices in LoRA are adjusted accordingly so that all PETs have similar sizes of parameters to store during inference.

GPT-2 Large: The hyper parameters used for GPT-2 Large are shown in Table 6. The bottleneck dimension of adapters is set as 16 in SST2 and 32 in MNLI, which results in around 0.19% trainable parameters in SST and 0.39% in MNLI for the PETs. For XSUM and MT tasks, a bottleneck dimension of 100 used for the adapters, around 1.19% trainable parameters are introduced. The rank of introduced matrices in LoRA are adjusted accordingly so that all PETs have similar sizes of parameters to store during inference.

The data processing procedure for GPT-2 follows the practice by Lin et al., 2020. For each task, we add task-related embedding at certain positions to help the model better identify the different parts of the inputs. The main difference is that we do not use the segment embedding (or token type embedding), as in our initial experiments we found that adding segment embedding didn't contribute

to the performance improvement. We list the data processing detail for different tasks in the following part.

A.1 SST2

We convert the classification task as a generation task. We directly generate the sentiment label (positive or negative) of the input. For example, the training sample: {**text**: a sometimes tedious film. **label**: 0 } is processed as {**input**: **<start>**a sometimes tedious film.**<sep>**, **label**: negative**<end>** }. Tokens in angle brackets (e.g. **<start>**, **<sep>** and **<end>**) are special tokens with trainable embeddings. The special token **<end>** is used as the stop generation token for all tasks. The maximum length of input is set as 128 tokens.

A.2 MNLI

We convert this task as a generation task. For the following train sample:

- **premise**: Everyone really likes the newest benefits
- **hypothesis**: The new rights are nice enough
- **label**: 2

We prepend the prefix text to the hypothesis and premise, and insert special task embedding at certain positions:

- **input**: **<start>***[hypothesis]*:Everyone really likes the newest benefits *[premise]*:The new rights are nice enough**<sep>**
- **label**: entailment**<end>**

where *[hypothesis]* and *[premise]* are plain text prefix, **<start>**, **<sep>** and **<end>** are special tokens. The maximum length of the input is set as 128 tokens.

A.3 E2E

For E2E generation task, we follow the practice by Lin et al., 2020. The following train sample:

- **meaning representation**: name: Alimentum, area: city centre, familyFriendly:no
- **human reference**: There is a place in the city centre, Alimentum, that is not family-friendly.

is processed as:

- **input**: **<start><name>**Alimentum**<area>**city centre**<familyFriendly>**no**<sep>**
- **label**: There is a place in the city centre, Alimentum, that is not family-friendly.**<end>**

	Learning Rate	Batch Size	Train Steps
SST2:			
Fine Tuning	5e-5	32	5 epochs
Prefix Tuning	1e-4	32	10 epochs
PA	1e-3	32	10 epochs
LoRA	5e-4	32	10 epochs
DoRA	1e-3	32	10 epochs
PrAd-PA	1e-3	32	10 epochs
PrAd-LoRA	1e-4	32	10 epochs
PrAd-DoRA	1e-3	32	10 epochs
MNLI:			
Fine Tuning	5e-5	32	5 epochs
Prefix Tuning	2e-3	32	5 epochs
PA	2e-3	32	5 epochs
LoRA	1e-4	32	5 epochs
PrAd-PA	2e-3	32	5 epochs
PrAd-LoRA	1e-4	32	5 epochs
E2E:			
Fine Tuning	5e-5	32	60000
Prefix Tuning	2e-3	32	60000
PA	2e-3	32	60000
LoRA	5e-4	32	60000
PrAd-PA	1e-3	32	60000
PrAd-LoRA	5e-4	32	60000
XSUM:			
Fine Tuning	5e-5	32	60000
Prefix Tuning	1e-3	32	60000
SA	5e-4	32	60000
PA	2e-3	32	60000
LoRA	5e-4	32	60000
DoRA	5e-4	32	60000
LoRA-FFN	5e-4	32	60000
PrAd-SA	5e-4	32	60000
PrAd-PA	2e-3	32	60000
PrAd-LoRA	5e-4	32	60000
PrAd-LoRA-FFN	5e-4	32	60000
PrAd-DoRA	5e-4	32	60000
WebNLG:			
Fine Tuning	5e-5	32	18000
Prefix Tuning	1e-3	32	18000
SA	1e-4	32	18000
PA	5e-4	32	18000
LoRA	5e-4	32	18000
PrAd-SA	5e-4	32	18000
PrAd-PA	1e-3	32	18000
PrAd-LoRA	5e-4	32	18000
MT:			
Fine Tuning	5e-5	32	60000
Prefix Tuning	2e-3	32	60000
SA	8e-4	32	60000
PA	1e-3	32	60000
LoRA	1e-3	32	60000
LoRA-FFN	1e-3	32	60000
PrAd-SA	1e-3	32	60000
PrAd-PA	3e-3	32	60000
PrAd-LoRA	6e-4	32	60000
PrAd-LoRA-FFN	1e-3	32	60000

Table 5: Hyper parameters settings for different tasks with GPT-2 Small.

	Learning Rate	Batch Size	Train Steps
SST2:			
PA	2e-4	16	5 epochs
LoRA	2e-4	16	5 epochs
PrAd-PA	5e-4	16	5 epochs
PrAd-LoRA	2e-4	16	5 epochs
MNLI:			
PA	2e-4	16	5 epochs
LoRA	8e-5	16	5 epochs
PrAd-PA	2e-4	16	5 epochs
PrAd-LoRA	8e-5	16	5 epochs
XSUM:			
PA	2e-4	8	5 epochs
LoRA	2e-4	8	5 epochs
PrAd-PA	2e-4	8	5 epochs
PrAd-LoRA	2e-4	8	5 epochs
MT:			
PA	2e-4	16	10 epochs
LoRA	5e-4	16	10 epochs
PrAd-PA	5e-4	16	10 epochs
PrAd-LoRA	2e-4	16	10 epochs

Table 6: Hyper parameters settings for different tasks with GPT-2 Large.

where tokens in angle brackets (e.g.**<start>**, **<name>** etc.) are special tokens. The maximum length of the input and label are both set as 256 tokens.

A.4 WebNLG

For the WebNLG task, the input consists of multiple triples. We also insert special tokens to identify the different parts and concatenate them as texts. For the example:

- **triples**: (United States, capital, Washington D.C.), (Albany Oregon, isPartOf, United States)
- **text**: Albany, Oregon is part of the United States, where Washington, D.C. is the capital.

is processes as:

- **input**: **<start>** **<subject>** United States **<property>** capital **<object>** Washington D.C. **<triple-sep>** **<subject>** Albany Oregon **<property>** isPartOf **<object>** United States **<sep>**
- **label**: Albany, Oregon is part of the United States, where Washington, D.C. is the capital.**<end>**

where tokens in angle brackets are special tokens. The maximum length of input and label are both set as 256 tokens.

A.5 XSUM and MT

The inputs of XSUM and MT are processed similarly. Suppose the task input is "[*task input*]" and the target sequence is "[*task output*]". We insert "**<start>**" at the begging and "**<sep>**" at end of the task input. We also append an "**<end>**" at the end of the target sequence. And we get the input-label pair as **input**: **<start>**[*task input*]**<sep>**, **label**: [*task output*]**<end>**. We set the maximum length as 100 tokens for both input and output in MT. In XSUM, the maximum length is 400 tokens for input and 130 for output.

B Experiment Details for LLaMA-7B

For experiments with LLaMA-7B, the experiments of SST2, MNLI and MT were ran on 4 NVIDIA A100 40G. The experiments of XSUM were ran on 4 NVIDIA A800 80G. We use greedy decoding during inference for all tasks. The bottleneck dimension of the adapters is set as 4 in SST2 and MNLI , which leads to around 0.031% trainable parameters (compared with the base model) in the PETs. For XSUM and MT, a bottleneck dimension of 32 is used for the adapters, which results in around 0.248% trainable parameters in the PETs. We use "huggingface transformers" and PyTorch packages for the implementation of different methods. We use deepspeed package for efficient training. For all tasks, we use warm-up steps of 500 and grad norm of 1. We use AdamW as the optimizer. Other hyper parameters used are shown in Table 7.

We use a simple data processing procedure for all the tasks. Suppose the task input is "[task input]" and the target sequence is "[task output]". The training sequence is formulated as "#INPUT: [task input] #OUTPUT: [task output]". For example, for task SST2, we have a task input: "*a sometimes tedious film*" and task output: "*negative*". The model needs to generate the label: "*negative*" conditioned on the text: "#INPUT: *a sometimes tedious film* #OUTPUT:". The maximum input length for SST2, MNLI, MT is set as 256 tokens. The maximum input length for XSUM is set as 512 tokens. The maximum output length for MT and XSUM is set as 256 tokens.

C Additional Experiment Results

C.1 Inference Time Comparison

This section presents supplementary experimental evaluations on inference time. The average inference time across 50 experimental runs was

	Learning Rate	Batch Size	Train Steps
SST2:			
PA	5e-5	32	3 epochs
LoRA	5e-5	32	3 epochs
PrAd-PA	5e-5	32	3 epochs
PrAd-LoRA	5e-5	32	3 epochs
MNLI:			
PA	2e-3	32	2 epochs
LoRA	1e-4	32	2 epochs
PrAd-PA	2e-3	32	2 epochs
PrAd-LoRA	1e-4	32	2 epochs
XSUM:			
PA	2e-4	32	3 epochs
LoRA	2e-4	32	3 epochs
PrAd-PA	2e-4	32	3 epochs
PrAd-LoRA	2e-4	32	3 epochs
MT:			
PA	2e-4	32	3 epochs
LoRA	2e-4	32	3 epochs
PrAd-PA	2e-4	32	3 epochs
PrAd-LoRA	2e-4	32	3 epochs

Table 7: Hyper parameters settings for different tasks with LLaMA-7B.

	SST2	**XSUM**
	Accuracy	ROUGE-1/2/L
DoRA	92.09	32.97/12.16/26.40
MaM	90.94	33.67/12.94/26.92
PrAd-DoRA	92.09	32.94/12.15/26.34
PrAd-MaM	90.48	33.91/13.04/27.20

Table 8: Performance of DoRA and MaM on task SST2, XSUM on GPT 2 Small.

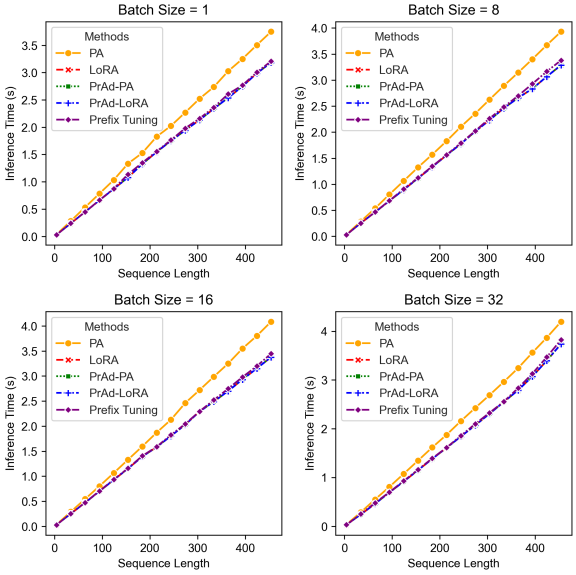


Figure 4: The inference latency (s) of different methods averaged over 50 runs on an NVIDIA A100 40G with GPT-2 Small. In each run, we encode an input of "Sequence Length" and generate an output sequence of the same length. The prefix length in Prefix Tuning and the bottleneck dimension in the adapters are set as 32.

measured for each method using a single NVIDIA A100 GPU with 40GB memory. In each run, we encode an input of "Sequence Length" and generate an output sequence of the same length. We insert parallel adapters only in the FFN layer. The increased latency of adapter-based methods compared with Fine Tuning on GPT-2 Small is depicted in Figure 3. The results indicate that PA can introduces 15% to 25 % additional inference latency, while PrAd-PA introduces negligible additional inference latency in most cases. Figure 5 presents the increased inference latency compared with Fine Tuning for GPT-2 Large. While PA introduces significant additional inference latency during short sequence and small batch generation, PrAd maintains nearly negligible latency impact under identical conditions. Figures 6 and 7 present the inference time of the adapter-based PETs at batch sizes of 1 and 16 respectively, providing empirical evidence for the superior efficiency of our approach when applied to large-scale models.

C.2 Experiments with DoRA and MaM

The flexibility of the framework allows for easy derivation of other PrAd variants. To further validate our approach, we perform additional experiments using MaM (He et al., 2021) and DoRA (Liu et al., 2024). MaM combines Prefix Tuning

	WebNLG								
	BLEU			MET			TER↓		
	S	U	A	S	U	A	S	U	A
Fine Tuning	$61.68_{.70}$	$27.67_{.37}$	$45.33_{.88}$	$0.444_{.002}$	$0.302_{.003}$	$0.375_{.002}$	$0.362_{.010}$	$0.737_{.017}$	$0.534_{.013}$
Prefix Tuning	$62.41_{.44}$	$\mathbf{40.79}_{.77}$	$\mathbf{52.95}_{.41}$	$0.444_{.002}$	$\mathbf{0.362}_{.002}$	$\underline{0.405}_{.001}$	$0.351_{.003}$	$\mathbf{0.550}_{.008}$	$\mathbf{0.442}_{.004}$
SA	$62.01_{.25}$	$\underline{38.27}_{.53}$	$51.36_{.41}$	$0.442_{.001}$	$0.351_{.002}$	$0.399_{.001}$	$0.357_{.004}$	$0.584_{.004}$	$0.461_{.004}$
PA	$\underline{63.28}_{.10}$	$36.08_{.50}$	$50.69_{.33}$	$\underline{0.450}_{.000}$	$0.350_{.001}$	$0.402_{.001}$	$\underline{0.346}_{.002}$	$0.605_{.009}$	$0.465_{.003}$
LoRA	$\mathbf{63.47}_{.22}$	$38.26_{.63}$	$\underline{52.07}_{.36}$	$\mathbf{0.453}_{.000}$	$0.353_{.001}$	$\mathbf{0.406}_{.001}$	$\mathbf{0.340}_{.002}$	$\underline{0.575}_{.001}$	$\underline{0.448}_{.006}$
PrAd-SA	$62.91_{.14}$	$38.05_{.58}$	$51.36_{.23}$	$0.445_{.001}$	$\underline{0.354}_{.004}$	$0.402_{.002}$	$0.357_{.003}$	$0.594_{.007}$	$0.465_{.004}$
PrAd-PA	$63.18_{.21}$	$36.91_{.14}$	$51.38_{.21}$	$0.446_{.001}$	$0.351_{.001}$	$0.400_{.001}$	$0.349_{.001}$	$0.592_{.008}$	$0.454_{.006}$
PrAd-LoRA	$62.85_{.25}$	$37.55_{.96}$	$51.72_{.64}$	$0.445_{.001}$	$0.345_{.002}$	$0.398_{.001}$	$0.347_{.002}$	$0.605_{.020}$	$0.465_{.009}$

Table 9: Performance on the task WebNLG with GPT-2 Small. "U" denotes unseen categories, "S" denotes seen categories, and "A" indicates all categories in the test set of WebNLG.

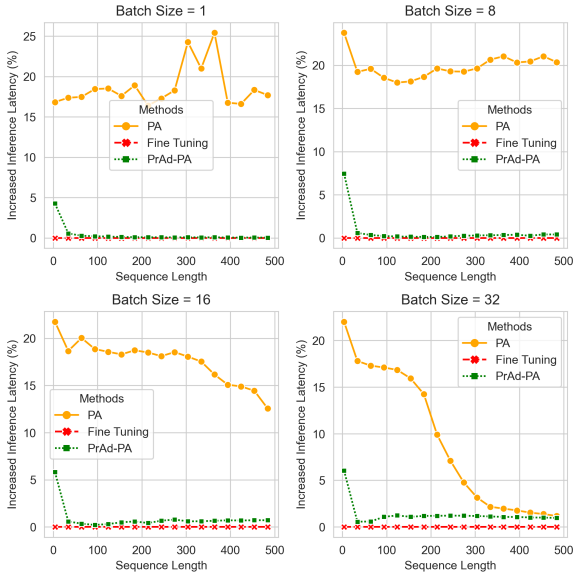


Figure 5: The increased inference latency (%) compared with Fine Tuning on GPT-2 Large. The bottleneck dimension in the adapters are set as 16.

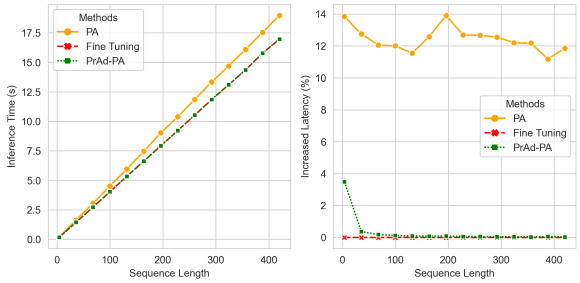


Figure 6: The inference latency of adapter-based methods compared with Fine Tuning on LLaMA-7B. The batch size is set as 1. The bottleneck dimension in the adapters are set as 4.

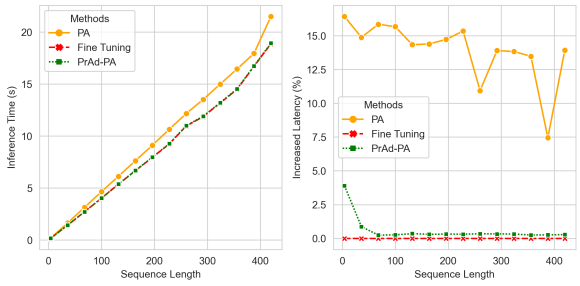


Figure 7: The inference latency of adapter-based methods compared with Fine Tuning on LLaMA-7B. The batch size is set as 16. The bottleneck dimension in the adapters are set as 4.

and PA and leverages the strengths of both. DoRA is an enhanced version of LoRA, demonstrating superior performance across specific task domains. Conventional DoRA and MaM use adapters both in the prefill phase and decode phase. By applying the adapters only in the prefill phase, we develop PrAd-DoRA and PrAd-MaM. We insert adapters only in the FFN layer with bottleneck dimension set as 4 for SST2 and 40 for XSUM. The prefix length is set 8 for SST2 and 30 for XSUM in MaM and PrAd-MaM. The experimental results in Table 8 indicate that our methods achieve comparable performance to existing approaches, demonstrating their effectiveness.

C.3 Performance on WebNLG

Table 9 presents the detailed results on WebNLG.