Ahead-of-Time P-Tuning

Anonymous ACL submission

Abstract

This paper proposes a simple reparametrization for Prefix-Tuning – AoT P-Tuning, in which we embed prefixes to a hidden state before evaluating the attention mechanism, saving a considerable amount of time needed for evaluation.

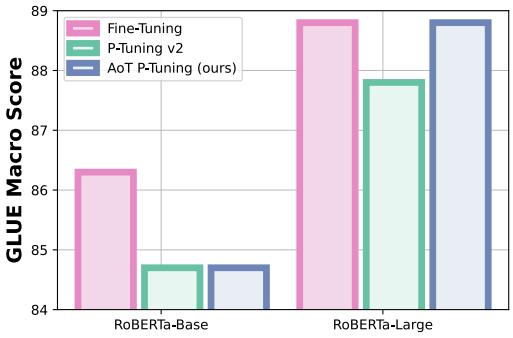
We experimented with the proposed method on GLUE Benchmarking Datasets and observed that AoT P-tuning performed on par with or better than P-Tuning v2 while being up to 1.3× times faster during inference.

1 Introduction

P-Tuning (Liu et al., 2021b,a; Lester et al., 2021) is a promising way to fine-tune large Language Models (LMs) (Devlin et al., 2019; Lan et al., 2020; Liu et al., 2019; Radford et al., 2019). While it currently underperforms compared to other methods for parameter-efficient fine-tuning (Hu et al., 2022; Houlsby et al., 2019) on a wide range of tasks (Ding et al., 2022), it has a practical, valuable property that allows it to evaluate different trained prompts parallel in a multi-task manner. This property is why researchers aim to further develop P-Tuning methods.

Although it is possible to perform multi-task evaluation with P-Tuning, it introduces significant computational overhead due to the concatenation of prefixes to sequences and the evaluation of attention mechanism (Vaswani et al., 2017) on longer sequences.

We propose a simple mechanism for parameter-efficient fine-tuning of Language Models, namely **Ahead-of-Time (AoT) P-Tuning**, for which we add input-dependent bias before each Transformer layer. Same as P-Tuning, it is possible to use AoT P-Tuning in multi-task inference setups when a single backbone LM is used for several downstream tasks.



(a)

Figure 1: GLUE Macro scores (higher is better) across RoBERTa-Base and RoBERTa-Large backbone models with plain Fine-Tuning, P-Tuning v2, and proposed AoT P-Tuning (with FC reparametrization). With the Base model, AoT P-Tuning reached the same result as P-Tuning v2. While for RoBERTa-Large, AoT P-Tuning outperformed P-Tuning v2 and reached the same score as Fine-Tuning. See Section 5.2 for more details.

The contribution of this paper can be summarized as follows:

1. We described the intuition behind AoT P-Tuning, which illustrates the connection of the proposed method with P-Tuning.

2. We proposed two reparameterizations of AoT P-Tuning weights: first based on a factorized matrix trained from scratch, and second based on a LM's embeddings matrix passed through a trainable Fully Connected network.

3. We experimented with the proposed method on GLUE Benchmarking Datasets (Wang et al., 2018) with the RoBERTa (Liu et al., 2019) model and observed that AoT P-Tuning performed on par with or better than P-Tuning v2 (Liu et al., 2021a) while being up to 1.3× times faster during evaluation.

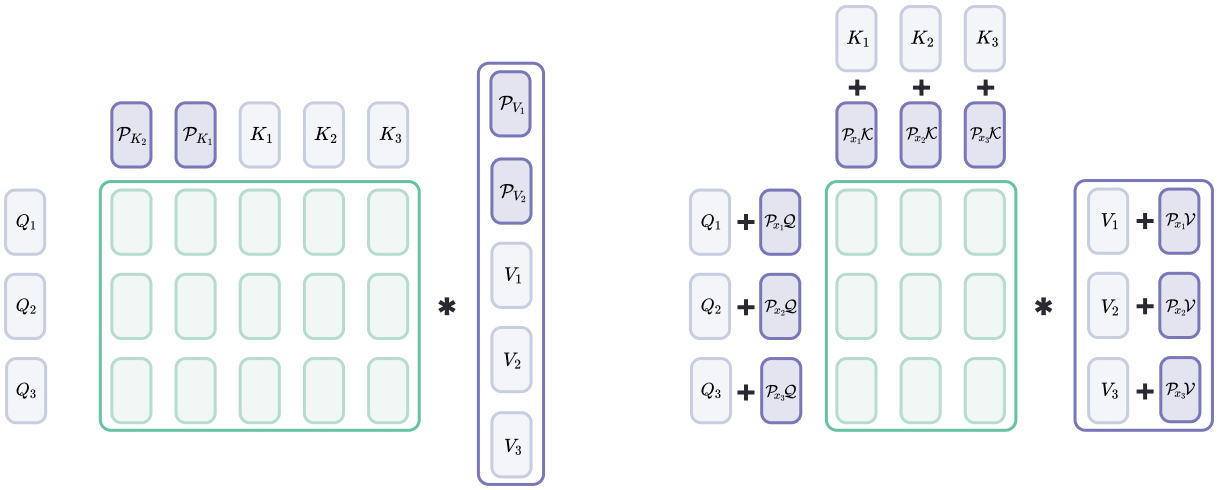


Figure 2: Schematic comparison of P-Tuning v2 (left), and AoT P-Tuning (right). While plain P-Tuning concatenates soft prompts to the sequences and thus causes computational overhead, AoT P-Tuning directly adds input-dependent biases to Q, K, and V matrices. See Section 4 for more details of AoT P-Tuning architecture. Since the sequence length is not increased, AoT P-Tuning takes significantly less time to evaluate, only requiring the overhead of adding biases to the input sequence (See Section 5.3 for experiments with inference speed).

2 Recent Works

Currently, a wide range of different methods could be referenced with P-Tuning. Liu et al. (2021b) proposed to add soft prompts to embeddings of the input sequence of the GPT-2 model (Radford et al., 2019) to train it on classification tasks. Lester et al. (2021) proposed a scheme similar to the one used in Liu et al. (2021b) but trained a T5 model (Raffel et al., 2020) with P-Tuning to show how the performance of the method changes with the increased scale of the backbone model.

Lately, Qin and Eisner (2021); Li and Liang (2021); Liu et al. (2021a) proposed to add prefixes not only to the input embeddings but also at each layer of the Transformer model. Also, Liu et al. (2021a) proposed training linear classification head on top of the backbone model instead of utilizing LM head to obtain classification results.

Due to this range of similar methods, we will follow the naming used by Liu et al. (2021a) and refer to Prompt-Tuning (adding soft prompts to the input embeddings) as P-Tuning v1 and to Prefix-Tuning (adding soft prefixes at each layer of Transformer backbone) as P-Tuning v2.

3 Background

3.1 Transformer Evaluation

Having an input sequence $x = \{x_1, \dots, x_n\}$, where x_i is token index, the embeddings of input texts are evaluated as $H^0 = \{E_{x_1}, \dots, E_{x_n}\}$, where $E \in \mathbb{R}^{|V| \times d}$ is the embeddings matrix, $|V|$ is the vocabulary size, d is the size of the hidden state of the model, and E_{x_i} is an embedding of token x_i. Then hidden states H^i are passed to the $(i+1)$-th layer of the Transformer to evaluate H^{i+1}, with total l number of layers.

To do so, H^i are firstly mapped through three matrices $\mathcal{Q}$, $\mathcal{K}$, $\mathcal{V} \in \mathbb{R}^{d \times d}$ to get Q, K and V, which are then used to evaluate the result of attention layer as:

$$A = \text{attention}(Q, K, V) = \\ = \text{softmax}(\frac{QK^T}{\sqrt{d}})V \in \mathbb{R}^{n \times d}. \quad (1)$$

After evaluation of A, it is passed through the remaining layers[1], including residual connections and FC layers to get H^{i+1}. We omit layer index i for attention result A for visibility here and later.

3.2 P-Tuning v1

Having a pre-trained Transformer LM with parameters Θ, instead of fine-tuning all parameters of this model on a downstream task, it is possible to define soft prompts $\mathcal{P} \in \mathbb{R}^{p \times d}$(Liu et al., 2021b), where p is the length of prompt. $\mathcal{P}$ is then concatenated to embeddings of input sequence as:

$$H'^0 = \text{concat}(\mathcal{P}, H^0) \in \mathbb{R}^{(p+n) \times d}. \quad (2)$$

[1]In fact, Transformer architecture implies evaluation of multi-head Attention. We omit this fact in this paper for simplification since all derivations could be easily extended on the multi-head case.

Then, only $\mathcal{P}$ and Classification Head are fine-tuned on a downstream task, while Θ remains frozen[2].

3.3 P-Tuning v2

Instead of adding a single prompt $\mathcal{P}$ to the X, Liu et al. (2021a) proposed to add soft prefixes at each layer of the Transformer model. To apply P-Tuning v2, soft prefixes $\mathcal{P}_K, \mathcal{P}_V \in \mathbb{R}^{p \times d}$ are defined for each layer and concatenated to the K and V matrices before evaluating the attention $K' = \text{concat}(\mathcal{P}_K, K)$, $V' = \text{concat}(\mathcal{P}_V, V)$. Then, Attention is evaluated as follows:

$$A' = \text{attention}(Q, K', V'). \tag{3}$$

i-th component of A' could be then written as:

$$A'_i = \sum_{j=1}^{p} a_j(Q_i, K')\mathcal{P}_{V_j} + \sum_{k=1}^{n} a_{k+p}(Q_i, K')V_k. \tag{4}$$

Note that $a \in \mathbb{R}^{p+n}$ are attention weights and thus $\sum_{j=1}^{p+n} a_j = 1$.

As for P-Tuning v1, only parameters of soft prefixes $\mathcal{P}_K, \mathcal{P}_V$ and Classification Head are optimized on a downstream task while freezing parameters of a backbone model.

3.4 P-Tuning Overhead

While the Transformer model has $\mathbb{O}(n^2)$ time complexity and GPU memory consumption for sequence length n. For P-Tuning v1, this complexity transforms into $\mathbb{O}((n+p)^2)$ since the length of input sequence is increased by the length of the prompt p, while for P-Tuning v2 the complexity is equal to $\mathbb{O}(n(n+p))$.

Liu et al. (2021a) showed that for some tasks, prompt length p could reach values of 100, increasing time and memory footprints during the evaluation.

4 Ahead-of-Time P-Tuning

4.1 Proposed Mechanism

With AoT P-Tuning, we propose to augment each Transformer layer with a simple procedure. We define trainable matrices $\mathcal{P} \in \mathbb{R}^{|V| \times d}$ for each layer. Then before evaluation of the i-th layer, we modify hidden states as follows

$$H'^i = H^i + \{\mathcal{P}_{x_1}, \ldots, \mathcal{P}_{x_n}\} \in \mathbb{R}^{n \times d}, \tag{5}$$

where $\mathcal{P}_{x_j} \in \mathbb{R}^d$ is a lockup of x_j-th prompt embedding from $\mathcal{P}$. Such a scheme allows us to save a significant amount of time during evaluation since AoT P-Tuning does not imply an increase in sequence length. Note that AoT P-Tuning, same as plain P-Tuning, could be evaluated in parallel with several tasks in a batch due to the fact that performing look-up from $\mathcal{P}$ can be easily parallelized.

As for P-Tuning v1 and P-Tuning v2, we only optimize parameters of $\mathcal{P}$ and Classification Head during fine-tuning.

4.2 AoT P-Tuning Parameter Efficiency

It is notable that, in most cases, one cannot afford to optimize the weight $\mathcal{P} \in \mathbb{R}^{|V| \times d}$ for each layer. If we consider training RoBERTa-Large with such a scheme (which has $|V| = 50265$, $d = 1024$ and $l = 24$), then storing all biases $\mathcal{P}$ will exceed 1.2B parameters, while the model itself has roughly 350M parameters.

To overcome this limitation, we propose two reparametrizations of $\mathcal{P}$ so that it can use fewer parameters during training.

The first is based on the Kronecker product (namely, **Kronecker AoT P-Tuning**). More specifically, we reparametrize $\mathcal{P}$ as

$$\mathcal{P} = (\mathcal{A} \otimes \mathcal{B})\mathcal{C}, \tag{6}$$

where $\mathcal{A} \in \mathbb{R}^{a \times r}$, $\mathcal{B} \in \mathbb{R}^{b \times r}$, $\mathcal{C} \in \mathbb{R}^{r^2 \times d}$, a and b are selected in such a way so $a * b = |V|$, r is a rank of factorization which is a hyperparameter to tune, and $\otimes$ denotes the Kronecker product.

With this reparametrization, training AoT P-Tuning becomes tractable. E.g., for RoBERTa-Large, with $a = 256$, $b = 200$, and $r = 20$, $\mathcal{P}$ will contain roughly 10M parameters, which is less than 3% of the total number of parameters in the model[3].

The second approach to work with $\mathcal{P}$, which we used in our experiments, is based on passing the

[2]Original implementation of P-Tuning v1 (Liu et al., 2021b) implied utilizing the LM Head of a pre-trained model instead of training a Classification Head. However, Liu et al. (2021a) later showed that using a separate Classification Head performs marginally better.

[3]One may note that $256 * 200 = 51200 \neq 50265$. However, 50265 is hard to factorize efficiently since $50265 = 1117 * 3^2 * 5$. Because of this fact, we chose to mostly factorize $\mathcal{P}$ in such a way as to make it slightly larger than the original vocabulary size. Doing so allows us to select more appropriate a and b from the perspective of parameter and computational efficiency.

Model	$A'_i =$	Computatonal Complexity	Trainable Parameters
P-Tuning v2	$\sum_{j=1}^{p} a_j(Q_i, K')\mathcal{P}_{V_j} + \sum_{k=1}^{n} a_{k+p}(Q_i, K')V_k$	$\mathbb{O}(n(p+n))$	$2ldp$
Kron. AoT P-Tuning	$\sum_{j=1}^{n} a_j(Q'_i, K')\mathcal{P}_{x_j}\mathcal{V} + \sum_{j=1}^{n} a_j(Q'_i, K')V_j$	$\mathbb{O}(n^2)$	$l(2rd + r + d)$
FC AoT P-Tuning		$\mathbb{O}(n^2)$	$lr(a + b + rd)$

Table 1: Side-by-side comparison of P-Tuning v2 and AoT P-Tuning (See Sections 3.3, 4 for details). For implied attention results A'_i, differences in evaluation are shown in color.

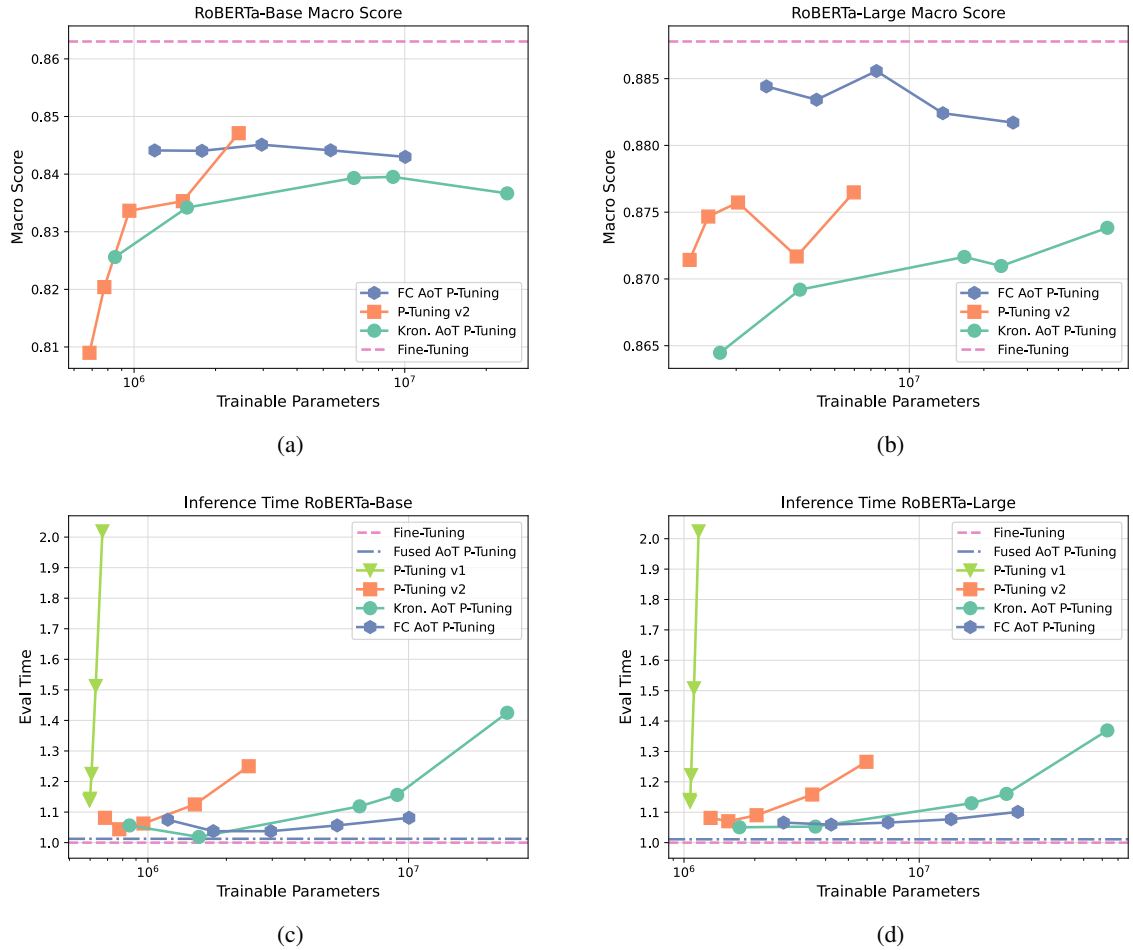


Figure 3: (a-b) GLUE macro scores for AoT P-Tuning, P-Tuning v1, and P-Tuning v2 with RoBERTa-Base and RoBERTa-Large. P-Tuning v2 reaches comparable with AoT P-Tuning results only with large prefix sizes. See Section 5.2 for details. (c-d) Comparison of AoT P-Tuning evaluation time with P-Tuning v1 and P-Tuning v2 for RoBERTa-Base and RoBERTa-Large. We evaluated AoT P-Tuning in two scenarios: with fused weight $\mathcal{P}$ and with the re-evaluation of $\mathcal{P}$ during the inference to reduce memory footprint (See Section 4.2 for more details). Fused AoT P-Tuning adds negligible computational overhead compared to plain Fine-Tuning and is faster than P-Tuning v2 up to $1.3\times$ times. Depending on prefix size, re-evaluating FC AoT P-Tuning performs up to $1.25\times$ times faster than P-Tuning v2. See Section 5.3 for more details.

embeddings matrix E through a learnable Fully Connected network (namely, **FC AoT P-Tuning**). Thus, we reparametrize $\mathcal{P}$ as

$$\mathcal{P} = f(E\mathcal{W}_1 + b_1)\mathcal{W}_2 + b_2, \tag{7}$$

where $W_1 \in \mathbb{R}^{d \times r}$, $b_1 \in \mathbb{R}^r$, $\mathcal{W}_2 \in \mathbb{R}^{r \times d}$, $b_2 \in \mathbb{R}^d$, f is a non-linearity, and r is a rank of mapping, which is also hyperparameter to tune as for Kronecker AoT P-Tuning.

With FC AoT P-Tuning, we utilize knowledge stored in the pre-trained embeddings matrix E, which should hypothetically perform better than training $\mathcal{P}$ from scratch as Kronecker AoT P-Tuning.

Note that for both Kronecker and FC AoT P-Tuning, we can evaluate only specific rows $\{\mathcal{P}_{x_i}, \dots, \mathcal{P}_{x_n}\}$ for input sequence $\{x_1, \dots, x_n\}$, making training more efficient.

For both reparametrizations, $\mathcal{P}$ could be fused once training is complete, and thus the rank of factorization r does not affect inference speed. During the evaluation, there is no need to store the full $\mathcal{P}$ in GPU memory. Instead, it could be stored in RAM, and only rows of these matrices should be placed in GPU memory to be added to the hidden states before each layer.

From a certain perspective, choosing between AoT P-Tuning and P-Tuning is a trade-off between evaluation speed and RAM consumption during inference. If RAM is limited, then usual P-Tuning could be used at the cost of slower inference. In other cases, AoT P-Tuning could be used if there is enough volume of RAM and inference speed is crucial. Although, in most cases, $\mathcal{P}$ matrices for different tasks could be easily stored in the RAM. For RoBERTa-Large, a single task parameter will roughly require 2.4Gb if stored in half-precision.

However, as we observed later in our experiments, for FC AoT P-Tuning, performing fusing is not crucial, and the re-evaluation of $\{\mathcal{P}_{x_i}, \dots, \mathcal{P}_{x_n}\}$ for each sequence ran at 98.5% the speed of fused $\mathcal{P}$ (See Section 5.3 for more details).

4.3 Intuition Behind AoT P-Tuning and Connection to P-Tuning

Having H', after passing through $\mathcal{Q}$, $\mathcal{K}$, and $\mathcal{V}$ we obtain Q', K', and V'. Note that $V' = H\mathcal{V} + \{\mathcal{P}_{x_1}, \dots, \mathcal{P}_{x_n}\}\mathcal{V} \stackrel{\text{def}}{=} V + \mathcal{P}_x\mathcal{V}$.

The result of the evaluation of Attention with AoT P-Tuning could be seen as:

$$A'_i = \text{attention}(Q', K', V') = \\ = \sum_{j=1}^{n} a_j(Q'_i, K')\mathcal{P}_{x_j}\mathcal{V} + \sum_{j=1}^{n} a_j(Q'_i, K')V_j. \tag{8}$$

From such a perspective, there is a clear connection of AoT P-Tuning (Equation 8) to P-Tuning v2 (Equation 4) with the following changes:

1. For AoT P-Tuning, attention weights a_j, $j \in \overline{1, l}$ are used for both terms in Equation 8.

2. For AoT, P-Tuning, attention is evaluated on modified Q'. In addition, there is a difference in the form of dependency of K' and V' on prefix weight. For AoT P-Tuning, we add prefixes to the K and V, while for P-Tuning v2 prefixes are concatenated to these matrices.

3. For AoT P-Tuning, the first term of Equation 8 implies evaluation of Attention with a prompt which is dependent on input text, while for P-Tuning v2, the prompt $\mathcal{P}_V$ is constant.

Considering Equation 8, AoT could be seen as a form of the P-Tuning method, for which we embed prefixes before evaluating the attention layer[4]. See Table 1 for the side-by-side comparison of AoT P-Tuning and P-Tuning v2.

5 GLUE

5.1 Experimental Details

We compared AoT P-Tuning (Kronecker and FC reparametrizations of $\mathcal{P}$) with P-Tuning v1, P-Tuning v2, and plain fine-tuning on GLUE Benchmarking Datasets (Wang et al., 2018). For each fine-tuning approach, we experimented with RoBERTa-Base and RoBERTa-Large backbone models.

For each task, we performed a grid hyperparameter search (see Appendix Table 4 for hyperparameter ranges[5].). Each hyperparameter set was evaluated with 5 different seed values. Each value we report for GLUE experiments is a median of task

[4]It is possible to think of AoT P-Tuning as a method which adds bias **after** the evaluation of Transformer layer. In this case, it could be seen as a method that directly models the result of the evaluation of P-Tuning v2 with a slightly different order of computations. However, we believe that this way is more difficult to consider.

[5]Note that we used a wider range for RTE task since we observed that P-Tuning v2 required larger hyperparameter assignments to reach results reported by Liu et al. (2021a)

RoBERTa-Base					
Model	STS-B	SST-2	RTE	QQP	
Fine-Tuning	90.6 ± 0.3	95.0 ± 0.2	81.2 ± 0.7	89.6 ± 0.2	
P-Tuning v1	86.9 ± 0.9	94.0 ± 0.3	60.3 ± 2.4	82.2 ± 1.5	
P-Tuning v2	89.2 ± 0.3	**94.6 ± 0.2**	**80.5 ± 3.4**	86.4 ± 3.3	
Kron. AoT P-Tuning (ours)	89.7 ± 0.2	94.0 ± 0.2	77.6 ± 1.4	88.2 ± 0.1	
FC AoT P-Tuning (ours)	**90.0 ± 0.2**	94.4 ± 0.3	78.0 ± 1.3	**87.9 ± 0.2**	
	QNLI	MRPC	MNLI	CoLA	Macro
Fine-Tuning	92.4 ± 0.1	90.8 ± 0.5	87.0 ± 0.3	63.8 ± 1.4	86.3
P-Tuning v1	88.3 ± 0.5	82.0 ± 1.7	80.8 ± 0.6	45.8 ± 27.1	77.5
P-Tuning v2	**91.9 ± 1.6**	89.1 ± 1.1	85.3 ± 0.2	**60.7 ± 2.6**	**84.7**
Kron. AoT P-Tuning (ours)	90.7 ± 0.4	89.5 ± 1.1	84.6 ± 0.1	59.3 ± 1.2	84.2
FC AoT P-Tuning (ours)	91.3 ± 0.4	**90.3 ± 0.3**	**85.4 ± 0.1**	60.3 ± 2.2	**84.7**
RoBERTa-Large					
Model	STS-B	SST-2	RTE	QQP	
Fine-Tuning	91.9 ± 0.2	96.1 ± 0.4	88.1 ± 1.5	90.3 ± 0.2	
P-Tuning v1	75.5 ± 6.3	94.4 ± 0.4	62.8 ± 2.3	76.9 ± 2.5	
P-Tuning v2	91.0 ± 0.4	96.1 ± 0.3	87.4 ± 1.5	86.6 ± 0.6	
Kron. AoT P-Tuning (ours)	91.1 ± 0.8	96.2 ± 0.2	84.8 ± 1.3	**89.4 ± 0.1**	
FC AoT P-Tuning (ours)	**91.7 ± 0.4**	**96.7 ± 0.1**	**88.4 ± 0.9**	88.7 ± 0.2	
	QNLI	MRPC	MNLI	CoLA	Macro
Fine-Tuning	94.3 ± 0.2	91.6 ± 0.6	89.9 ± 0.2	68.1 ± 1.9	88.8
P-Tuning v1	79.1 ± 2.4	79.0 ± 1.1	75.9 ± 18.3	24.7 ± 17.6	71.0
P-Tuning v2	94.0 ± 1.1	91.2 ± 0.9	89.4 ± 0.7	66.9 ± 1.5	87.8
Kron. AoT P-Tuning (ours)	**94.2 ± 0.1**	89.7 ± 0.9	89.3 ± 0.1	65.5 ± 1.9	87.5
FC AoT P-Tuning (ours)	94.1 ± 0.2	**91.6 ± 0.8**	**89.6 ± 0.1**	**69.2 ± 0.9**	**88.8**

Table 2: Results on the GLUE Dev set. Each result is median and std across several seeds, and the Macro column is a mean score across all tasks. Fine-tuning is omitted from comparison with other methods and was not bolded for visibility.

scores across different seeds. See Appendix Table 5 for a list of metrics used for each task.

We used Adam (Kingma and Ba, 2015) optimizer with a constant learning rate for each task and stopped training once the validation metric stopped increasing ("patience" parameter in Appendix Table 3).

For Kronecker AoT P-Tuning we parametrized matrix $\mathcal{P} = (\mathcal{A} \otimes \mathcal{B})\mathcal{C}$ with $a = 256$, and $b = 200$. $\mathcal{A}$ and $\mathcal{B}$ were initialized randomly, while $\mathcal{C}$ was initialized as a zero matrix. For FC AoT P-Tuning, we initialized $\mathcal{W}_1$ randomly, while $\mathcal{W}_2$, b_1, and b_2 were initialized with zeros. See Section 4.2 for reparametrization details.

Each experiment was run on a single NVidia A100 GPU with total compute time being roughly equal to 400 days.

5.2 Results

See Table 2 for the results of trained models.

We observed that FC AoT P-Tuning performed better than Kronecker AoT P-Tuning. We hypothesize that this result is mostly caused by the fact that FC reparametrization utilized a pre-trained embedding matrix rather than learning biases from scratch.

For RoBERTa-Base, FC AoT P-Tuning performed on par with P-Tuning v2 and produced the same Macro score. For RoBERTa-Large, FC AoT P-Tuning outperformed P-Tuning v2 and showed a Macro score equal to plain Fine-Tuning. We also observed that both AoT P-Tuning reparametriza-

tions showed a lower variance of metrics across different seeds. Note that P-Tuning v1 performed unstably and showed higher results with RoBERTa-Base (although still underperforming other methods by a large margin).

See Figures 3(a-b) for macro scores of P-Tuning v2 and AoT P-Tuning with different prefix lengths p and prefix ranks r[6]. We observed that P-Tuning v2 performed worse with shorter prompt lengths and became comparable to or better than AoT P-Tuning when $p > 50$. For RoBERTa-Large, FC AoT P-Tuning performed better for all prefixes r. We also provide per-task results with different prefix scales (see Appendix Figure 4). It is notable that P-Tuning v2, in most cases, suffers from small prefix size p and achieves results comparable with AoT P-Tuning with larger p (which corresponds to the result with Figures 3(a-b)).

With per-task Expected Validation Performance (EVP) (Dodge et al., 2019), we observed that P-Tuning v2 highly depends on the number of hyperparameter assignments for some tasks (see Figure 5). E.g. for the RTE dataset, P-Tuning v2 only outperformed Kronecker AoT P-Tuning with 100 hyperparameter assignments, which forced us to increase the ranges of hyperparameter search for this task (see Section 5.1). However, we observed that FC AoT P-Tuning outperformed other methods when a large number of hyperparameter assignments was reached.

5.3 Inference Speed Overhead

With Figures 3(c-d), we also investigated the computational overhead of AoT P-Tuning compared to other baselines.

To estimate inference speed overhead, we evaluated each model 100 times on a sequence with length $n = 128$ and batch size 256.

We evaluated AoT P-Tuning in two setups. The first setup implies fusing $\mathcal{P}$ for the inference, thus saving computational time at the cost of a higher memory footprint. Since $\mathcal{P}$ is fused, it no longer depends on factorization rank r for both FC and Kronecker AoT P-Tuning.

For the second setup, we did not fuse $\mathcal{P}$, but rather evaluated $\{\mathcal{P}_{x_i}, \dots, \mathcal{P}_{x_n}\}$ for each sequence. This approach emulates setup with limited memory during inference when fusing $\mathcal{P}$ is not feasible.

[6]Note that best macro result across different scales of prefixes in these Figures differs from the macro result from Table 2 since the macro score from Table 2 aggregates scores with different prefix scales.

The growth of p P-Tuning v1 quickly reaches $2\times$ speed overhead since its complexity depends on p quadratically. While P-Tuning v2 involves linear dependency on p (see Section 3.4 for details), it also reaches up to $1.3\times$ inference speed overhead for large prefix lengths p.

Fused AoT P-Tuning adds negligible computational overhead (less than 1%) compared to plain Fine-Tuning. Compared to P-Tuning v2, Fused AoT P-Tuning performed up to $1.3\times$ times faster, depending on the prefix size used for P-Tuning v2.

When $\mathcal{P}$ is not fused, FC AoT P-Tuning is performing $1.13 - 1.25\times$ times faster than P-Tuning v2 with large prefixes p, which indicates that for this reparametrization, **performing weight fusing is not crucial in most cases** and huge inference speed-up could be achieved without it. Although not performing fusing of $\mathcal{P}$ could reduce memory footprint during the inference, in such a setup, there is no ability to perform multi-task inference, which is available for both P-Tuning v1/v2 and Fused AoT P-Tuning.

Kronecker's reparametrization performed worse. For small factorization rates (e.g., $r \in [5, 10]$), it performed comparable to FC AoT P-Tuning. Although, for larger r values, it performed up to $1.12\times$ times slower than P-Tuning v2. Thus, it is important to fuse $\mathcal{P}$ with such a reparametrization when a large rank r is used.

6 Conclusion and Future Work

In this paper, we proposed a new method for parameter-efficient fine-tuning of pre-trained models: AoT P-Tuning. We also proposed two reparametrizations of learnable weights for this method.

We observed that AoT P-Tuning performed on par with P-Tuning v2 with RoBERTa-Base backbone while outperforming it and performing on par with plain Fine-Tuning with RoBERTa-Large measured by GLUE Benchmarking Datasets score.

While performing on par or better than P-Tuning v2, AoT P-Tuning performed up to $1.3\times$ times faster than P-Tuning v2, adding a negligible inference time footprint compared to plain Fine-Tuning. When FC AoT P-Tuning is used, we observed that one could not fuse weights $\mathcal{P}$ in order not to introduce memory footprint since it performs up to $1.25\times$ times faster than P-Tuning v2.

We experimented only with two reparametrizations based on the Kronecker product and FC net-

work. It is possible to further explore other possible reparametrizations for weight $\mathcal{P}$, which could further increase the performance of the proposed method.

References

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

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

Jesse Dodge, Suchin Gururangan, Dallas Card, Roy Schwartz, and Noah A. Smith. 2019. Show your work: Improved reporting of experimental results. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2185–2194, Hong Kong, China. Association for Computational Linguistics.

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 *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *ICLR 2022*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

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

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

Xiao Liu, Kaixuan Ji, Yicheng Fu, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021a. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *CoRR*, abs/2110.07602.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021b. Gpt understands, too. *arXiv:2103.10385*.

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

Guanghui Qin and Jason Eisner. 2021. Learning how to ask: Querying LMs with mixtures of soft prompts. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5203–5212, Online. Association for Computational Linguistics.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Alex Wang, Amapreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. Cite arxiv:1804.07461Comment: https://gluebenchmark.com/.

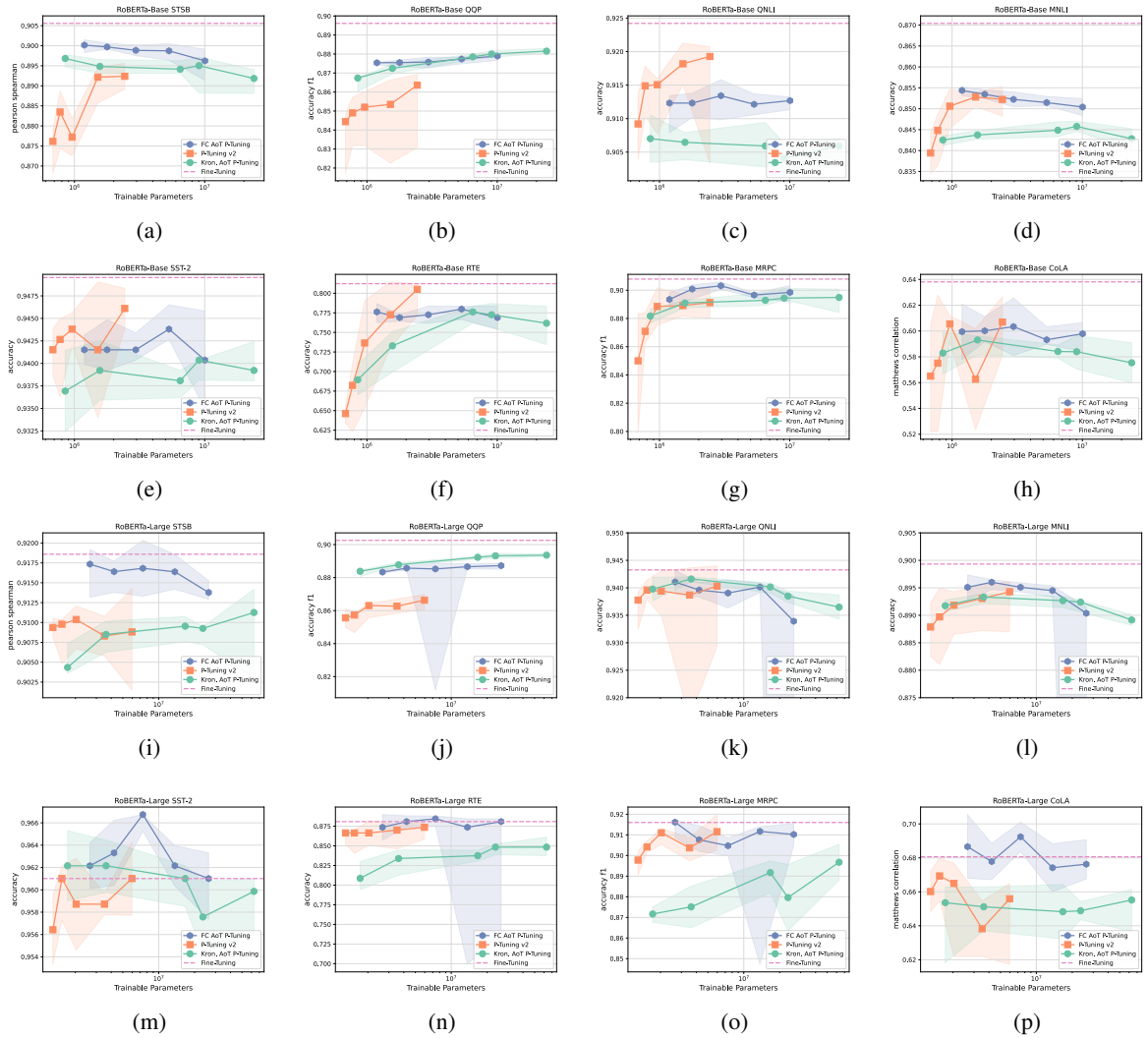


Figure 4: Per-task GLUE Benchmarking Dataset results for a different number of trained parameters of P-Tuning v2 and AoT P-Tuning with RoBERTa-Base (a-h) and RoBERTa-Large (i-p). We also provide results of plain fine-tuning for reference. See Section 5.2 for more details.

	RTE	**MNLI, QQP**	**QNLI**	**Other Tasks**
Epochs	200	5	10	100
Patience	20	2	2	10

Table 3: The number of maximum epochs used for each GLUE and SuperGLUE Task. Once the Dev score stopped increasing for "patience" steps, training was halted. See Section 5.1 for more details.

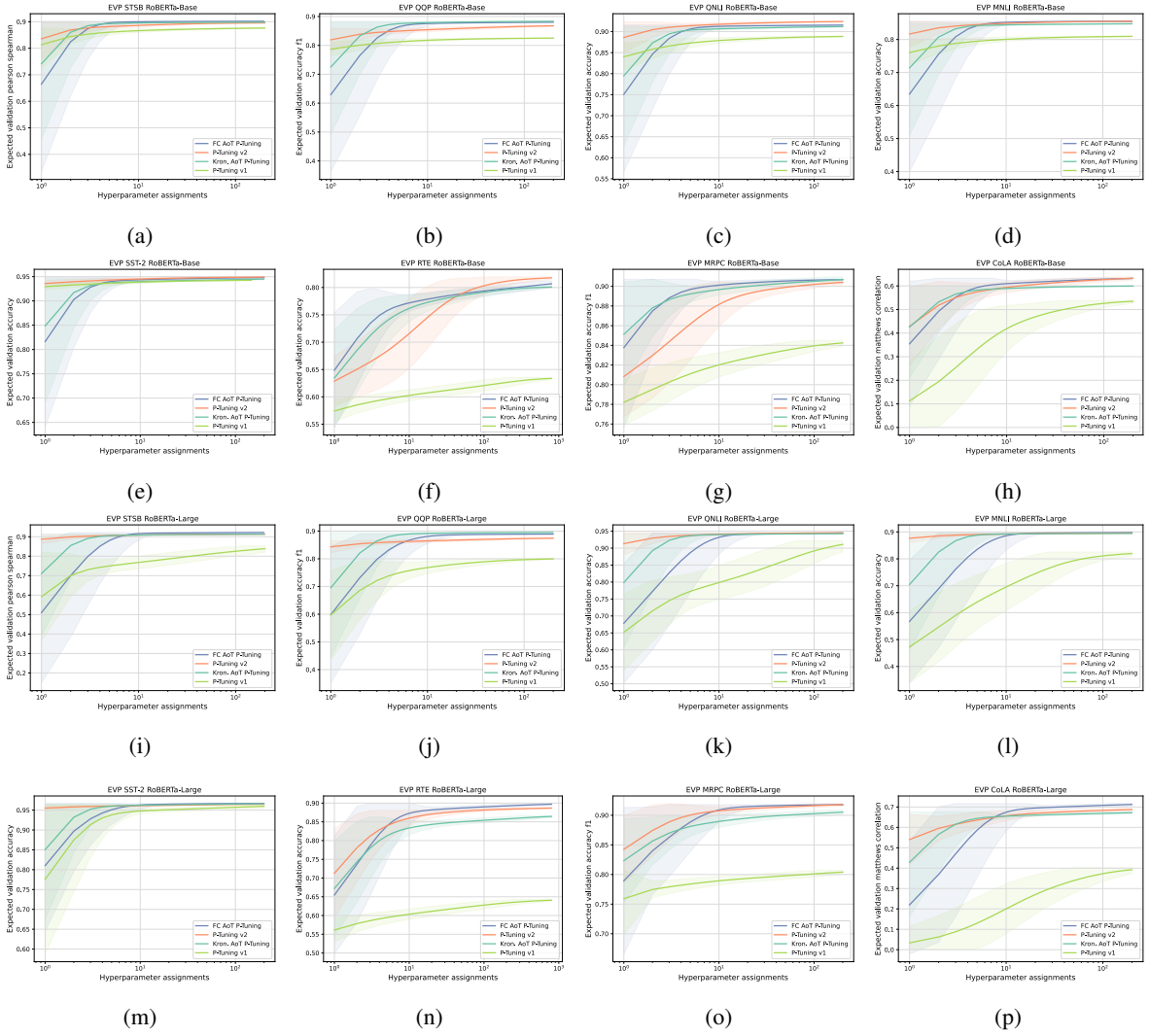


Figure 5: Expected Validation Performance (Dodge et al., 2019) of trained models with GLUE Benchmarking Datasets for RoBERTa-Base (a-h) and RoBERTa-Large (i-p). See Section 5.2 for more details.

Parameter	Range
All Tasks, except RTE	
P-Tuning v1/v2/AoT	
batch size	16, 64
learning rate	1e−4, 5e−4, 5e−3, 1e−3
p	5, 10, 20, 50, 100
Kron. r	5, 10, 25, 30, 50
FC r	32, 64, 128, 256, 512
Fine-Tuning	
learning rate	1e−5, 5e−5, 1e−4, 5e−4, 5e−3
RTE	
batch size	16, 32, 64, 128
learning rate	1e−5, 5e−5, 1e−4, 5e−4, 5e−3, 1e−3, 2e−3, 1e−2

Table 4: Hyperparameter ranges used in experiments with GLUE benchmarking datasets. p is the prompt length used for P-Tuning v1/v2, and r is the rank of weight factorization used for AoT P-Tuning (See Section 4.2). Each hyperparameter set was evaluated with different seed values. See Section 5.1 for more details.

Task	Metric
CoLA	Mattews Correlation
MRPC	$\frac{\text{Accuracy}+\text{F1}}{2}$
RTE	Accuracy
SST-2	Accuracy
MNLI	Accuracy
QNLI	Accuracy
QQP	$\frac{\text{Accuracy}+\text{F1}}{2}$
STSB	$\frac{\text{Pearson}+\text{Spearman}}{2}$

Table 5: Metrics used in our experiments for each task. See Section 5.1 for more details.