State-offset Tuning: State-based Parameter-Efficient Fine-Tuning for State Space Models

Wonjun Kang[1,2*] **Kevin Galim**[2*] **Yuchen Zeng**[3*] **Minjae Lee**[2]
Hyung Il Koo[2,4†] **Nam Ik Cho**[1]
[1] Seoul National University [2] FuriosaAI [3] UW-Madison [4] Ajou University
{kangwj1995, kevin.galim, minjae.lee, hikoo}@furiosa.ai, yzeng58@wisc.edu, nicho@snu.ac.kr

Abstract

State Space Models (SSMs) have emerged as efficient alternatives to Transformers, mitigating their quadratic computational cost. However, the application of Parameter-Efficient Fine-Tuning (PEFT) methods to SSMs remains largely unexplored. In particular, prompt-based methods like Prompt Tuning and Prefix-Tuning, which are widely used in Transformers, do not perform well on SSMs. To address this, we propose *state-based methods* as a superior alternative to prompt-based methods. This new family of methods naturally stems from the architectural characteristics of SSMs. State-based methods adjust state-related features directly instead of depending on external prompts. Furthermore, we introduce a novel state-based PEFT method: *State-offset Tuning*. At every timestep, our method directly affects the state at the current step, leading to more effective adaptation. Through extensive experiments across diverse datasets, we demonstrate the effectiveness of our method. Code is available at `https://github.com/furiosa-ai/ssm-state-tuning`.

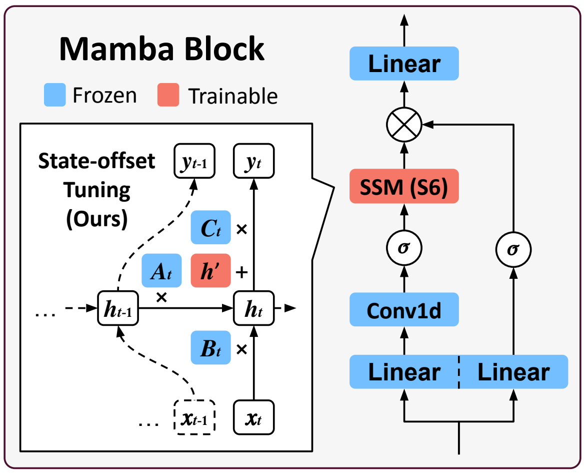


Figure 1: Illustration of our proposed State-offset Tuning on a Mamba block (Gu and Dao, 2024). State-offset Tuning injects a trainable state-offset $\boldsymbol{h}'$ at each timestep in the SSM module while keeping other parameters frozen, enabling parameter-efficient fine-tuning and improved downstream performance.

1 Introduction

Large Language Models (LLMs) have gained significant attention for their strong performance in NLP tasks (Achiam et al., 2023; Brown et al., 2020), but suffer from the quadratic complexity of Transformer architectures (Vaswani et al., 2017). To mitigate this, subquadratic alternatives have gained interest (Katharopoulos et al., 2020; Peng et al., 2023; Sun et al., 2023), with State Space Models (SSMs) emerging as a promising solution (Gu and Dao, 2024; Dao and Gu, 2024).

Meanwhile, as LLMs scale up, full fine-tuning for downstream tasks becomes prohibitively expensive. Consequently, Parameter-Efficient Fine-Tuning (PEFT) (Houlsby et al., 2019; Hu et al., 2021; He et al., 2021; Zaken et al., 2022; Liu et al., 2021, 2022; Zeng and Lee, 2024) has emerged, which aims to reduce the number of trainable parameters while achieving adaptation performance comparable to full fine-tuning.

However, research on PEFT methods for SSMs remains limited despite their growing popularity. For instance, prompt-based PEFT methods, such as Prompt Tuning (Lester et al., 2021) and Prefix-Tuning (Li and Liang, 2021), have been widely applied to Transformers but fail to adapt effectively to SSMs (Galim et al., 2024). Therefore, new PEFT strategies tailored to SSMs are needed to fully leverage their architectural properties.

To bridge this gap, we introduce *state-based PEFT methods* that leverage the intrinsic properties of SSMs, offering a superior alternative to prompt-based methods. Building on this concept, we propose *State-offset Tuning*. This method directly ad-

[*]Equal contribution.
[†]Corresponding author.

]

justs the state-related features rather than relying on external prompts, enabling more effective adaptation.

In summary, our main contributions are:

- We introduce *state-based methods*, a new family of PEFT techniques for SSMs, offering a superior alternative to prompt-based approaches.
- We propose *State-offset Tuning* as a new state-based PEFT method.
- We demonstrate the effectiveness of our method through experiments on a variety of datasets, consistently outperforming existing fine-tuning techniques.

2 Related Works

2.1 State Space Models

Linear State-Space Layers (LSSL) are one of the earliest applications of SSMs in sequence modeling (Gu et al., 2021), leveraging HiPPO (Gu et al., 2020) to initialize the state matrix. However, its high computational overhead limits practicality. Gu et al. (2022) introduced Structured State Space Models (S4), which mitigate this by structuring the state matrix. Recently, Mamba (Gu and Dao, 2024; Dao and Gu, 2024) enhanced modeling capabilities by introducing an input-dependent S6 block.

2.2 Parameter-Efficient Fine-Tuning

In this section, we review existing PEFT methods. For more details, see Sec. D.

Parameter-based Methods One approach to parameter-based PEFT methods is to selectively fine-tune specific layers within the model while keeping the remaining layers frozen. BitFit (Zaken et al., 2022) is a lightweight and effective strategy that focuses solely on fine-tuning a model's bias terms. Furthermore, LoRA (Hu et al., 2021) represents a notable parameter-based PEFT method by introducing low-rank matrices for weight updates, facilitating efficient adaptation.

Prompt-based Methods Instead of fine-tuning model parameters, Prompt Tuning (Lester et al., 2021) enhances models by prepending trainable soft embeddings to the prompt. Prefix-Tuning (Li and Liang, 2021) builds on this approach by injecting trainable embeddings into each Transformer layer, achieving strong adaptation results for Transformer-based LLMs.

PEFT for SSMs Concurrently, Galim et al. (2024) showed that LoRA outperforms prompt-based methods on SSMs. Furthermore, they proposed Selective Dimension Tuning (SDT) for fine-tuning the SSM module while applying LoRA on the linear projection matrices when fine-tuning Mamba models. Yoshimura et al. (2025) suggested a new PEFT method called Additional-scan, which increases the hidden state dimension of SSMs, fine-tuning only its additional parameters.

3 PEFT Methods on SSMs

SSM Preliminaries Assuming a single channel dimension, SSMs such as S4 (Gu et al., 2022) transform a signal $x_t \in \mathbb{R}$ into $y_t \in \mathbb{R}$ through an H-dimensional latent state $\boldsymbol{h}_t \in \mathbb{R}^H$ as below:

$$\boldsymbol{h}_t = \overline{\boldsymbol{A}}\boldsymbol{h}_{t-1} + \overline{\boldsymbol{B}}x_t, \qquad y_t = \boldsymbol{C}\boldsymbol{h}_t,$$

where $\overline{\boldsymbol{B}} \in \mathbb{R}^{H\times 1}$ controls input influence, $\overline{\boldsymbol{A}} \in \mathbb{R}^{H\times H}$ governs state dynamics, and $\boldsymbol{C} \in \mathbb{R}^{1\times H}$ maps the state to the output. $\overline{\boldsymbol{A}}$ and $\overline{\boldsymbol{B}}$ represent discretized versions of $\boldsymbol{A}$ and $\boldsymbol{B}$, parameterized by a learnable step size $\Delta \in \mathbb{R}$.

In S6 (the SSM module of Mamba), input dependency is integrated by using input-dependent $\overline{\boldsymbol{A}}_t$, $\overline{\boldsymbol{B}}_t$, and $\boldsymbol{C}_t$ at every timestep. Specifically, given D channels with $\boldsymbol{x}_t \in \mathbb{R}^D$, learnable parameters $\boldsymbol{W_B}, \boldsymbol{W_C} \in \mathbb{R}^{H\times D}$, and $\boldsymbol{W_\Delta} \in \mathbb{R}^{D\times D}$ compute $\boldsymbol{B}_t = \boldsymbol{W_B}\boldsymbol{x}_t$, $\boldsymbol{C}_t = \boldsymbol{W_C}\boldsymbol{x}_t$, and $\boldsymbol{\Delta} = \boldsymbol{W_\Delta}\boldsymbol{x}_t$. In this section, we consider S4 for simplicity.

3.1 Prompt-based PEFT Methods on SSMs

Prefix-Tuning Can Update Only the Initial State of an SSM Generally, SSMs assume that the initial hidden state is $\boldsymbol{h}_0 = \boldsymbol{0}$. We can express $\boldsymbol{h}_t$ with $\boldsymbol{h}_0$ as $\boldsymbol{h}_t = \sum_{i=1}^{t} \overline{\boldsymbol{A}}^{t-i}\overline{\boldsymbol{B}}_i x_i + \overline{\boldsymbol{A}}^t \boldsymbol{h}_0$.

Assume we have virtual tokens $x_{(-V+1)}, \dots, x_0$. If we prepend virtual tokens as prefix to the input sequence, we can write the updated $\widehat{\boldsymbol{h}}_t$ as below:

$$\widehat{\boldsymbol{h}}_t = \boldsymbol{h}_t + \overline{\boldsymbol{A}}^t \sum_{i=0}^{V-1} \overline{\boldsymbol{A}}^i \overline{\boldsymbol{B}} x_{-i} = \boldsymbol{h}_t + \overline{\boldsymbol{A}}^t \boldsymbol{h}_{\text{prefix}}.$$

By introducing a non-zero $\widehat{\boldsymbol{h}}_0$, we can substitute $\widehat{\boldsymbol{h}}_0$ for $\boldsymbol{h}_{\text{prefix}}$ making Prefix-Tuning, or optimizing virtual tokens, equivalent to updating the initial state. As optimized virtual tokens only affect the initial state $\widehat{\boldsymbol{h}}_0$, Prefix-Tuning's expressivity is upper-bounded by updating the initial state directly (Galim et al., 2024). Since Prefix-Tuning is an extended version of Prompt Tuning, this upper bound is applicable to Prompt Tuning as well.

Galim et al. (2024) showed Initial State Tuning, an advanced version of Prefix-Tuning, which directly optimizes the channel-specific initial state $\boldsymbol{h}' \in \mathbb{R}^H$, resulting in DH trainable parameters in total across all D channels. The updated output $\widehat{y}_t$ for Initial State Tuning can be written as in Table 1.

3.2 State-based Methods: A New Family of PEFT Methods for SSMs

We define state-based methods as a new family of PEFT methods specifically designed for SSMs. These methods directly modify the intrinsic state-related features within the SSM module.

In contrast, prompt-based methods, such as Prefix-Tuning, influence the hidden state of the SSM module indirectly by introducing external virtual tokens. While both approaches adjust the hidden state of the SSM module, state-based methods operate within the SSM module itself, offering a more direct and expressive adaptation strategy.

Based on our definition, we classify Initial State Tuning as a state-based method. While Initial State Tuning surpasses Prefix-Tuning (Galim et al., 2024), it still falls short compared to other fine-tuning methods on SSMs. To bridge this gap, we propose a novel state-based method for enhanced performance.

Initial State Tuning	$\widehat{y}_t = y_t + \boldsymbol{C}_t(\prod_{i=1}^t \overline{\boldsymbol{A}}_i)\boldsymbol{h}'$
State-offset Tuning ($\boldsymbol{h}$)	$\widehat{y}_t = y_t + \boldsymbol{C}_t\boldsymbol{h}'$
State-offset Tuning (y)	$\widehat{y}_t = y_t + y'$

Table 1: State-based methods for S6. Our methods eliminate the time-dependent coefficient $\prod_{i=1}^t \overline{\boldsymbol{A}}_i$, ensuring a uniform effect across timesteps.

Prompt-based	Timestep T	Timestep $T+1$
Prefix	$[\textbf{prefix}, x_1, \ldots, x_T] \rightarrow$	$[\textbf{prefix}, x_1, \ldots, x_T, \boldsymbol{x}_{T+1}]$
Suffix	$[x_1, \ldots, x_T, \textbf{suffix}] \rightarrow$	$[x_1, \ldots, x_T, \textbf{suffix}, x_{T+1}]$
Iterative Suffix	$[x_1, \ldots, x_T, \textbf{suffix}] \rightarrow$	$[x_1, \ldots, x_T, x_{T+1}, \textbf{suffix}]$

Table 2: Comparison of Prefix-Tuning, Suffix-Tuning, and Iterative Suffix-Tuning.

4 Proposed State-based PEFT Method

In this section, we propose *State-offset Tuning* as a new state-based PEFT method. A visual comparison with Initial State Tuning and Prefix-Tuning is provided in Sec. A.

4.1 State-offset Tuning

Initial State Tuning introduces an additional term $\boldsymbol{h}'$ with a coefficient $\overline{\boldsymbol{A}}^t$ for S4 and $\prod_{i=1}^t \overline{\boldsymbol{A}}_i$ for S6. However, this coefficient, which varies for each timestep, tends to decrease over time, leading to inconsistent effects. This is related to the issue that SSMs struggle to recall early tokens (Fu et al., 2022). To address this and ensure a consistent effect for each timestep, we introduce *State-offset Tuning*, which eliminates this coefficient.

State-offset Tuning adds a constant, learnable state-offset $\boldsymbol{h}'$ to the hidden state $\boldsymbol{h}$ before obtaining the updated output $\widehat{y}_t$ (Fig. 1). Therefore, unlike Initial State Tuning, State-offset Tuning does not alter the hidden state dynamics directly. Instead, State-offset Tuning adds a constant $\boldsymbol{h}'$ repetitively for each timestep, ensuring a uniform impact.

We formulate State-offset Tuning ($\boldsymbol{h}$) for S6 in Table 1, where we optimize $\boldsymbol{h}' \in \mathbb{R}^H$. In S4, $\boldsymbol{C}_t$ does not depend on the input, simplifying to a constant $\boldsymbol{C}$. This allows us to optimize a bias y' instead of $\boldsymbol{h}'$ (with $y' := \boldsymbol{C}\boldsymbol{h}'$ for each dimension). We name this method State-offset Tuning (y). For S4, State-offset Tuning (y) and State-offset Tuning ($\boldsymbol{h}$) are equivalent. In S6, opting for the simpler State-offset Tuning (y) enhances parameter efficiency by decreasing the tunable parameters from DH to D.

4.2 Connection to Prompt-based Methods

To further validate the methodology of State-offset Tuning, we examine its connection to prompt-based methods and demonstrate its correspondence to *Iterative Suffix-Tuning*.

Iterative Suffix-Tuning Li and Liang (2021) showed that in Transformers, inserting virtual tokens at the beginning (Prefix-Tuning) or the end (Suffix-Tuning, referred to as Infix-Tuning in their work) yields similar performance.

However, for SSMs, the position of the inserted virtual tokens is crucial, as these models tend to forget early tokens. The effect of Prefix-Tuning and Suffix-Tuning diminishes as the model processes subsequent timesteps. This leads to the question: how can we maintain consistent influence of virtual tokens across all timesteps in SSMs?

To achieve this, we propose Iterative Suffix-Tuning. As shown in Table 2, both Prefix-Tuning and Suffix-Tuning hold virtual tokens in fixed positions throughout all timesteps. Conversely, Iterative Suffix-Tuning shifts virtual tokens to the sequence's last position at each timestep, ensur-

Model Size		Mamba 1.4B									Mamba 130M			
	Dataset	Params	Spider					SAMSum			Params	DART		GLUE
Type	Method	(%)	All	Easy	Medium	Hard	Extra	R1	R2	RL	(%)	MET.	BLEU	Avg.
-	Pretrained	0.00	0.0	0.0	0.0	0.0	0.0	10.9	1.5	10.2	0.00	18.1	1.2	41.0
	Full Fine-tuning (All)	100.00	66.2	84.3	69.5	53.4	43.4	51.2	27.3	42.9	100.00	71.0	51.8	80.5
	Full Fine-tuning (S6)	4.46	56.7	76.6	57.8	46.0	34.9	51.1	26.9	42.2	4.31	70.3	48.7	79.3
Parameter based	LoRA	0.46	56.3	75.0	56.5	**50.6**	**33.7**	50.5	26.4	42.2	0.92	69.9	**50.8**	78.3
	BitFit	0.03	51.3	74.2	50.9	43.1	26.5	50.3	25.7	41.9	0.06	67.0	43.7	77.9
	Additional-scan	0.34	26.9	44.4	25.6	21.3	10.2	37.6	17.5	30.9	0.68	60.6	15.8	62.4
Prompt based	Prompt Tuning	0.01	43.6	65.3	42.4	33.3	25.3	50.1	25.6	41.6	0.04	66.2	39.8	63.8
	Prefix-Tuning	12.81	39.7	65.7	38.6	31.0	15.1	50.6	**26.5**	42.1	22.69	66.6	42.5	68.6
State based	Initial State Tuning	0.23	51.8	**77.8**	51.1	35.1	32.5	50.0	26.0	41.3	0.45	69.1	46.2	77.4
	State-offset Tuning (h)	0.23	**57.4**	77.4	**59.9**	44.8	**33.7**	**50.9**	**26.5**	**42.4**	0.45	**70.0**	47.0	**78.5**
	State-offset Tuning (y)	0.01	53.0	77.4	55.4	40.8	22.9	50.6	26.1	42.0	0.03	66.8	45.2	77.7

Table 3: Experimental results for fine-tuning the SSM module (S6) of Mamba (Gu and Dao, 2024) models. We assess Spider and its subsets using execution accuracy, SAMSum with ROUGE-1/2/L scores, DART using METEOR and BLEU scores, and GLUE by calculating the average score. To demonstrate the effectiveness of our methods, we configure the hyperparameters of each method to ensure their parameter budget is comparable to or exceeds that of our methods. **Bold** and underline indicate the best and the second-best results, respectively, among all methods (excluding full fine-tuning). Our State-offset Tuning ($\boldsymbol{h}$) outperforms all other methods on most datasets, and our State-offset Tuning (y) shows comparable or better performance than other methods despite its significantly fewer trainable parameters.

ing uniform influence in SSMs. This method is akin to how State-offset Tuning eliminates the time-varying coefficient in Initial State Tuning, enforcing a consistent effect at every timestep. We show that Iterative Suffix-Tuning in SSMs is equivalent to State-offset Tuning (as detailed in Sec. B).

5 Experiments

5.1 Experiment Setup

We conduct experiments for fine-tuning the SSM module (S6) using pretrained Mamba (Gu and Dao, 2024) and Mamba-2 (Dao and Gu, 2024) models on four datasets: Spider (Yu et al., 2018), SAMSum (Gliwa et al., 2019), DART (Nan et al., 2021), and GLUE (Wang et al., 2019). For further information on datasets, evaluation metrics, and experimental details, refer to Secs. E and F. We use LoRA (Hu et al., 2021), BitFit (Zaken et al., 2022), and Additional-scan (Yoshimura et al., 2025) as parameter-based methods. For prompt-based methods, we employ Prompt Tuning (Lester et al., 2021) and Prefix-Tuning[1] (Li and Liang, 2021). For state-based methods, we utilize Initial State Tuning (Galim et al., 2024), along with our proposed methods, State-offset Tuning ($\boldsymbol{h}$) and State-offset Tuning (y).

[1]Following Yoshimura et al. (2025), our Prefix-Tuning implementation for SSMs corresponds to their Affix-Tuning formulation, adapted specifically for SSM architectures.

5.2 Experimental Results

Table 3 shows the results on Mamba models. Additional results, including Mamba-2 results, are provided in Sec. G. In the appendix, we further compare the training speed, training memory usage, and computational overhead during inference between LoRA and State-offset Tuning ($\boldsymbol{h}$). Our findings show that State-offset Tuning ($\boldsymbol{h}$) is faster, more memory-efficient, and introduces lower FLOP overhead compared to LoRA. Additionally, we evaluate the performance of State-offset Tuning ($\boldsymbol{h}$) within SSMs against Prefix-Tuning in Transformers, further highlighting the effectiveness of our approach.

State-based Methods Outperform Prompt-based Methods Table 3 shows that all state-based methods outperform prompt-based methods, supporting the claim that state-based methods are superior to prompt-based methods on SSMs.

In particular, our State-offset Tuning ($\boldsymbol{h}$) achieves the best results among all tested PEFT methods on most datasets. Our State-offset Tuning (y) outperforms Initial State Tuning on most datasets, using just 0.01% of the parameters compared to 0.23% by Initial State Tuning.

State-offset Tuning Outperforms Parameter-Based Methods State-offset Tuning ($\boldsymbol{h}$) outperforms BitFit across all datasets and surpasses LoRA on most datasets. Notably, it also outperforms Additional-scan, a method specifically designed for fine-tuning SSM modules, across all datasets.

Furthermore, State-offset Tuning ($\boldsymbol{h}$) achieves performance comparable to full fine-tuning (S6), highlighting the effectiveness of state-based PEFT for SSM modules, despite using significantly fewer parameters. The results from Mamba-2 (Table 11) further validate the effectiveness of our method. We also include a comparison to Selective Dimension Tuning (SDT) (Galim et al., 2024) in Sec. G.4, showing that our method outperforms SDT while using fewer parameters.

6 Conclusion

In this paper, we introduce *state-based methods* as a new family of PEFT methods for State Space Models, serving as a superior alternative to prompt-based methods. We propose *State-offset Tuning* as a new state-based PEFT method and demonstrate its effectiveness through extensive experiments.

7 Limitations

While we demonstrate that State-offset Tuning is effective for fine-tuning SSMs in the text domain, its applicability to other domains, such as vision or speech, remains unexplored. Existing PEFT methods, such as LoRA and Prompt Tuning, have been successfully applied across various domains (Jia et al., 2022; Gal et al., 2023; Ran et al., 2024). Extending State-offset Tuning to models in other domains, such as Vision Mamba (Zhu et al., 2025), is an interesting direction for future work.

Potential Risks Our approach enables parameter-efficient fine-tuning (PEFT) of pretrained SSMs, significantly reducing the computational cost of adaptation. While this is beneficial for resource-constrained scenarios, it also presents potential risks. Specifically, adversaries could leverage our method to efficiently fine-tune pretrained SSMs on harmful or biased data, enabling the rapid adaptation of models for malicious purposes with minimal computational resources. This could lead to the proliferation of harmful or deceptive models that reinforce misinformation, bias, or toxicity. To mitigate these risks, future work should explore more robust safety measures, such as integrating ethical fine-tuning constraints and monitoring mechanisms.

References

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

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, Ann Arbor, Michigan. Association for Computational Linguistics.

Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second pascal recognising textual entailment challenge. In Proceedings of the second PASCAL challenges workshop on recognising textual entailment, volume 1. Citeseer.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. TAC, 7(8):1.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In International Conference on Machine Learning, pages 2397–2430. PMLR.

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. In Advances in Neural Information Processing Systems, volume 33, pages 1877–1901.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In Machine learning challenges workshop, pages 177–190. Springer.

Tri Dao and Albert Gu. 2024. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In International Conference on Machine Learning.

Bill Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In Third international workshop on paraphrasing (IWP2005).

Daniel Y Fu, Tri Dao, Khaled Kamal Saab, Armin W Thomas, Atri Rudra, and Christopher Re. 2022. Hungry hungry hippos: Towards language modeling with state space models. In International Conference on Learning Representations.

Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit Haim Bermano, Gal Chechik, and Daniel Cohen-Or. 2023. An image is worth one word: Personalizing text-to-image generation using textual inversion. In The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net.

Kevin Galim, Wonjun Kang, Yuchen Zeng, Hyung Il Koo, and Kangwook Lee. 2024. Parameter-efficient fine-tuning of state space models. arXiv preprint arXiv:2410.09016.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The Pile: An 800GB dataset of diverse text for language modeling. arXiv preprint arXiv:2101.00027.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and William B Dolan. 2007. The third pascal recognizing textual entailment challenge. In Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing, pages 1–9.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. EMNLP-IJCNLP 2019, page 70.

Albert Gu and Tri Dao. 2024. Mamba: Linear-time sequence modeling with selective state spaces. In First Conference on Language Modeling.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. 2020. Hippo: Recurrent memory with optimal polynomial projections. In Advances in Neural Information Processing Systems, volume 33, pages 1474–1487.

Albert Gu, Karan Goel, and Christopher Re. 2022. Efficiently modeling long sequences with structured state spaces. In International Conference on Learning Representations.

Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. 2021. Combining recurrent, convolutional, and continuous-time models with linear state space layers. In Advances in Neural Information Processing Systems, volume 34, pages 572–585.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. In International Conference on Learning Representations.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In International Conference on Machine Learning, pages 2790–2799.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. LoRA: Low-rank adaptation of large language models. In International Conference on Learning Representations.

Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. 2022. Visual prompt tuning. In European Conference on Computer Vision, pages 709–727. Springer.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are RNNs: Fast autoregressive transformers with linear attention. In International Conference on Machine Learning, pages 5156–5165.

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.

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

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

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-Tuning: Prompt Tuning Can Be Comparable to Fine-tuning Across Scales and Tasks. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 61–68.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. GPT Understands, Too. arXiv:2103.10385.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. Peft: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`.

Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, et al. 2021. DART: Open-Domain Structured Data Record to Text Generation. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 432–447.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. pages 311–318.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. Preprint, arXiv:1912.01703.

Bo Peng, Eric Alcaide, Quentin Gregory Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Nguyen Chung, Leon Derczynski, et al. 2023. RWKV: Reinventing RNNs for the transformer era. In The 2023 Conference on Empirical Methods in Natural Language Processing.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

Lingmin Ran, Xiaodong Cun, Jia-Wei Liu, Rui Zhao, Song Zijie, Xintao Wang, Jussi Keppo, and Mike Zheng Shou. 2024. X-adapter: Adding universal compatibility of plugins for upgraded diffusion model. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 8775–8784.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. 2023. S-lora: Serving thousands of concurrent lora adapters. CoRR.

Daniel G. A. Smith and Johnnie Gray. 2018. opt_einsum - a python package for optimizing contraction order for einsum-like expressions. Journal of Open Source Software, 3(26):753.

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.

Vladislav Sovrasov. 2018-2024. ptflops: a flops counting tool for neural networks in pytorch framework.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. 2023. Retentive network: A successor to transformer for large language models. arXiv preprint arXiv:2307.08621.

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, volume 30.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In International Conference on Learning Representations.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2019. Neural network acceptability judgments. Transactions of the Association for Computational Linguistics, 7:625–641.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL HLT 2018, pages 1112–1122. Association for Computational Linguistics (ACL).

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 38–45, Online. Association for Computational Linguistics.

Masakazu Yoshimura, Teruaki Hayashi, and Yota Maeda. 2025. MambaPEFT: Exploring parameter-efficient fine-tuning for mamba. In The Thirteenth International Conference on Learning Representations.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 3911–3921.

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

Yuchen Zeng and Kangwook Lee. 2024. The expressive power of low-rank adaptation. In International Conference on Learning Representations.

Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. 2025. Vision mamba: Efficient visual representation learning with bidirectional state space model. In Forty-first International Conference on Machine Learning.

A Visual Comparison of Prompt-based Methods and State-based Methods

Fig. 2 compares prompt-based methods and state-based methods, including our proposed State-offset Tuning, within the S6 block.

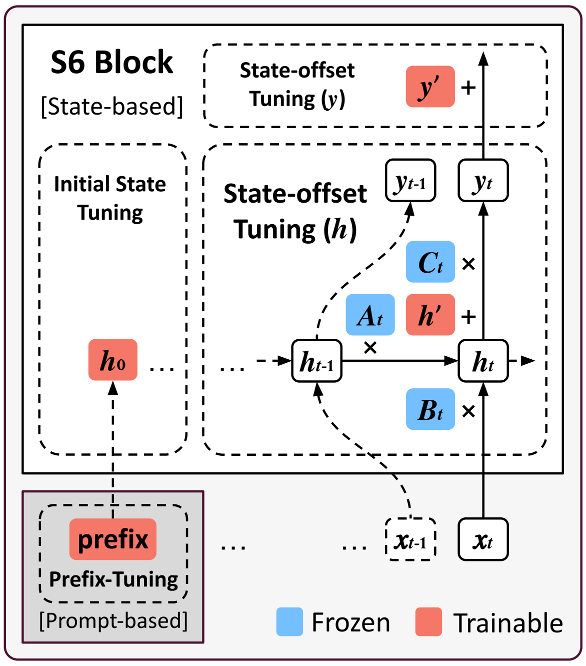


Figure 2: Visual comparison of prompt-based methods and state-based methods in the S6 block.

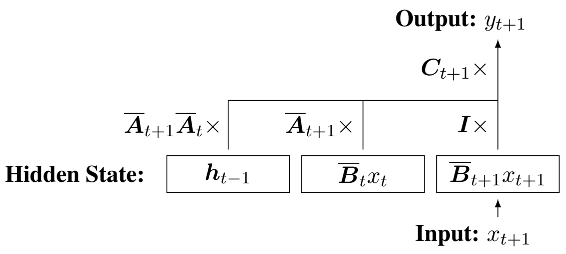
State-based Methods Operate within the SSM Module Fig. 2 shows that prompt-based methods, such as Prefix-Tuning, rely on virtual tokens external to the S6 block. In contrast, state-based methods, such as Initial State Tuning, State-offset Tuning ($\boldsymbol{h}$), and State-offset Tuning (y), directly adjust state-related features within the S6 block.

State-offset Tuning Affects the Current Timestep Figure 2 illustrates how Prefix-Tuning and Initial State Tuning modify features at early timesteps, indirectly affecting the current state. However, this impact diminishes over time. In contrast, State-offset Tuning ($\boldsymbol{h}$) and State-offset Tuning (y) directly influence the state at each timestep, resulting in more effective adaptation.

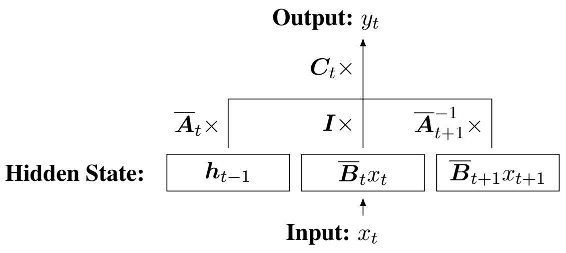
B Iterative Suffix-Tuning and State-offset Tuning

In this section, we show that Iterative Suffix-Tuning for SSMs is equivalent to State-offset Tuning.

State-offset Tuning is Iterative Suffix-Tuning Fig. 3 provides two different implementations of


Output: y_{t+1}
$\boldsymbol{C}_{t+1}\times$
$\overline{\boldsymbol{A}}_{t+1}\overline{\boldsymbol{A}}_t\times$
$\overline{\boldsymbol{A}}_{t+1}\times$
$\boldsymbol{I}\times$
Hidden State:
$\boldsymbol{h}_{t-1}$
$\overline{\boldsymbol{B}}_t x_t$
$\overline{\boldsymbol{B}}_{t+1}x_{t+1}$
Input: x_{t+1}

(a) Iterative Suffix-Tuning (with $t+1$ as current timestep)


Output: y_t
$\boldsymbol{C}_t\times$
$\overline{\boldsymbol{A}}_t\times$
$\boldsymbol{I}\times$
$\overline{\boldsymbol{A}}_{t+1}^{-1}\times$
Hidden State:
$\boldsymbol{h}_{t-1}$
$\overline{\boldsymbol{B}}_t x_t$
$\overline{\boldsymbol{B}}_{t+1}x_{t+1}$
Input: x_t

(b) Iterative Suffix-Tuning (with t as current timestep)

Figure 3: Two different implementations of Iterative Suffix-Tuning in S6. We show that Fig. 3b is equivalent to State-offset Tuning.

Iterative Suffix-Tuning on SSMs (S6) with virtual token (suffix) x_{t+1}. Fig. 3a views $t+1$ as current timestep. In this case, input-dependent $\boldsymbol{C}_{t+1} = \boldsymbol{W}_{\boldsymbol{C}}\boldsymbol{x}_{t+1}$ is determined solely by the suffix $\boldsymbol{x}_{t+1} \in \mathbb{R}^D$, which is constant at inference time, thus the input dependency of $\boldsymbol{C}$ is lost, reducing the expressive power of S6.

To address this, we view t as current timestep instead and interpret x_{t+1} as future token (Fig. 3b). Consequently, we time-shift x_{t+1} by multiplying it with the inverse of $\overline{\boldsymbol{A}}_{t+1}$.

$$\text{Fig. 3a:}\quad y_{t+1} = \boldsymbol{C}_{t+1}(\overline{\boldsymbol{A}}_{t+1}\boldsymbol{h}_t + \overline{\boldsymbol{B}}_{t+1}x_{t+1}),$$
$$\text{Fig. 3b:}\quad y_t \quad = \boldsymbol{C}_t(\boldsymbol{h}_t + \overline{\boldsymbol{A}}_{t+1}^{-1}\overline{\boldsymbol{B}}_{t+1}x_{t+1}).$$

Therefore, according to the equation corresponding to Fig. 3b, Iterative Suffix-Tuning can be implemented by updating only $\overline{\boldsymbol{A}}_{t+1}^{-1}\overline{\boldsymbol{B}}_{t+1}x_{t+1}$. Since this term depends solely on the constant suffix x_{t+1}, we can directly replace it with a learnable parameter $\boldsymbol{h}'$ ($\boldsymbol{h}' := \overline{\boldsymbol{A}}_{t+1}^{-1}\overline{\boldsymbol{B}}_{t+1}x_{t+1}$), which is equivelant to State-offset Tuning ($\boldsymbol{h}$) (Table 1).

C Low-Rank State-offset Tuning

State-offset Tuning ($\boldsymbol{h}$) shows superior parameter efficiency on Mamba versus other PEFT methods. To further reduce trainable parameters, we can represent the learnable state-offset as a product of two low-rank matrices, inspired by LoRA (Hu et al., 2021). This is particularly useful for Mamba-2, where the state dimension is larger than in Mamba, leading to an increased number of trainable parameters. In such cases, low-rank techniques can effectively mitigate the parameter overhead. Experimental results of State-offset Tuning ($\boldsymbol{h}$) with lower rank on Mamba-2 are provided in Sec. G.2.

D PEFT Baselines

In this section, we provide a more detailed description of the baseline methods.

LoRA (Hu et al., 2021) LoRA aims to fine-tune large models by maintaining the bulk of pretrained parameters untouched while introducing trainable low-rank matrices within each Transformer's layer. This method leverages linear algebra principles where a large matrix can be effectively approximated by two low-rank matrices, thus reducing the number of parameters. LoRA includes a scaling parameter to adjust the influence of original and LoRA weights during training. We use the Hugging Face version (Apache License 2.0, Mangrulkar et al. (2022)) of LoRA for our experiments.

Prompt Tuning (Lester et al., 2021) This method involves freezing the entire model and adding a trainable soft prompt to the input. The prompt consists of continuous virtual tokens that provide additional context.

Prefix-Tuning (Li and Liang, 2021) Similar to Prompt Tuning, Prefix-Tuning adds trainable tokens but extends them across every Transformer layer by appending trainable embeddings to the attention matrices. To combat the instability in training these prefixes, an over-parameterized MLP is utilized, which can be discarded after training.

BitFit (Zaken et al., 2022) This PEFT method simplifies fine-tuning by training only the bias terms while freezing the other model weights, drastically reducing trainable parameters.

SDT (Galim et al., 2024) SDT (Selective Dimension Tuning) employs a sparse updating approach for the matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ ($\boldsymbol{W_B}$ and $\boldsymbol{W_C}$ for S6), while additionally applying LoRA to the linear projection layers. All remaining layers are kept frozen. The process for determining which parameters to update involves a warmup stage, during which parameters are flagged as updatable if they exhibit a significant gradient magnitude. In our SDT experiments, we excluded LoRA from the linear projection layers and focused solely on its S6 component.

Additional-scan (Yoshimura et al., 2025) This approach enhances the model's expressivity by expanding the state dimensions for $\boldsymbol{A}$, $\boldsymbol{W_B}$, and $\boldsymbol{W_C}$. During training, only the added dimensions are marked as trainable.

E Datasets

Dataset	#Train	#Valid	#Epochs	Model size	Metrics
RTE	2490	277	10	130m	Acc.
MRPC	3668	408	10	130m	Acc.
CoLA	8551	1043	10	130m	Acc.
SST-2	67349	872	10	130m	Acc.
QNLI	104743	5463	10	130m	Acc.
QQP	363846	40430	3	130m	Acc.
MNLI	392702	19647	3	130m	Acc.
Spider	6918	1034	10	1.4B, 2.8B	Acc.
SAMSum	14732	819	10	1.4B	ROUGE
DART	62659	2768	10	130m	METEOR, BLEU

Table 4: Dataset details. We report the number of training and validation samples, number of training epochs, employed model size and evaluation metrics.

This paper examines four datasets across two domains: Natural Language Understanding (NLU) and Natural Language Generation (NLG). Table 4 presents detailed information for each dataset.

GLUE (Wang et al., 2019) A benchmark comprising nine tasks in English for assessing language understanding models, including sentiment analysis, linguistic acceptability, and question answering. We use the the following datasets: RTE (Dagan et al., 2005; Bar-Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009), MRPC (Dolan and Brockett, 2005), CoLA (Warstadt et al., 2019), SST-2 (Socher et al., 2013), QNLI (Rajpurkar et al., 2018), QQP[2], and MNLI (Williams et al., 2018). Evaluation is mainly through accuracy, except for CoLA where Matthews correlation is used. The final metric is calculated as the average accuracy (Matthews correlation for CoLA) across all datasets. The individual datasets are available under different permissive licenses. We use the version hosted at `https://huggingface.co/datasets/nyu-mll/glue`.

SAMSum (Gliwa et al., 2019) A dataset for dialogue summarization featuring about 16,000 synthetic conversations in English with summaries, created to simulate digital communications with

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

varied tones and styles. Its structure helps in developing systems that process conversational text. The dataset is evaluated via ROUGE score. This dataset is available under the CC BY-NC-ND 4.0 license. We use the version hosted at `https://huggingface.co/datasets/Samsung/samsum`.

Spider (Yu et al., 2018) A text-to-SQL dataset with 10,000 annotated SQL queries across 200+ databases, classifying queries from easy to extra hard based on SQL operation complexity. It involves translating English questions to SQL, evaluated via execution accuracy. Execution accuracy considers the output correct if the model's predicted SQL query and the ground truth SQL query yield the same results when executed on the database. This dataset is available under the CC BY-SA 4.0 license. We use the version hosted at `https://huggingface.co/datasets/xlangai/spider`.

DART (Nan et al., 2021) Comprising over 80,000 instances, DART focuses on English RDF-to-text generation, organized by structured data triples and corresponding text summaries. It is assessed using METEOR and BLEU metrics. This dataset is available under the MIT license. We use the version hosted at `https://huggingface.co/datasets/Yale-LILY/dart`.

F Experimental Details

For every dataset, we select the model size based on how difficult the dataset is and conduct a brief grid search for one epoch using a subset of the data (1k-2k instances) with learning rates of $\{4 \times 10^{-1}, 2 \times 10^{-1}, 1 \times 10^{-1}, ..., 1 \times 10^{-5}\}$. The best learning rate is then selected as the rate that has the lowest training loss. In our experimental results, we report the metric from the best epoch observed on the validation set during training, employing early stopping. Each experiment is conducted once. We apply fine-tuning methods to the SSM module (S6) of Mamba (130M, 1.4B, 2.8B)[3] and the SSM module (SSD) of Mamba-2 (130M, 1.3B)[4] pretrained from Pile (MIT License, Gao et al. (2020)) using AdamW (Loshchilov and Hutter, 2019) with a linear decay schedule for the learning rate. In general, we choose hyperparameters for each individual method to ensure that all methods operate within a similar parameter budget. Tables 5 and 6 show selected learning rates and chosen hyperparameters for each method. For assessing NLG tasks, we utilize beam search with five beams and a maximum beam length of 1024. BLEU (Papineni et al., 2002), ROUGE (Lin, 2004), and METEOR (Banerjee and Lavie, 2005) metrics are computed using Hugging Face's evaluate library[5].

We use an NVIDIA RTX 3090 24GB for training models with less than 1 billion parameters, and an NVIDIA H100 80GB for larger models. We implemented our project in PyTorch (Modified BSD license, Paszke et al. (2019)), utilizing the Hugging Face trainer (Apache License 2.0, Wolf et al. (2020)). We train with batch size 4 for 10 epochs on all datasets except QQP and MNLI for which we use 3 epochs, allowing each training run to finish in under 16 hours. This project spanned three months, utilizing four NVIDIA RTX 3090 24GB GPUs and four NVIDIA H100 80GB GPUs, totaling approximately 17,000 GPU hours.

G Additional Experimental Results

G.1 Mamba Results

Training Speed and Memory Usage We conduct a small experiment to compare the memory usage and training speed of State-offset Tuning ($\boldsymbol{h}$) and LoRA, as they performed most similarly in terms of dataset metrics in our experiments. Using a single H100 GPU, we train for 100 batch iterations with a batch size of 4 and a 1K context, continuously measuring memory usage and batch latency.

Table 7 shows the training speed and maximum memory usage for different Mamba sizes for State-offset Tuning ($\boldsymbol{h}$) and LoRA. State-offset Tuning ($\boldsymbol{h}$) uses less memory and is faster, even with more trainable parameters. In this experiment, we selected hyperparameters to ensure LoRA has less trainable parameters than State-offset Tuning ($\boldsymbol{h}$). We believe State-offset Tuning ($\boldsymbol{h}$)'s efficiency stems from our optimized einsum implementation, enhanced with the opt_einsum (MIT License, Smith and Gray (2018)) Python package to reduce memory usage and improve latency.

[3]Apache License 2.0, https://huggingface.co/state-spaces/mamba-{130m,1.4b,2.8b}

[4]Apache License 2.0, https://huggingface.co/state-spaces/mamba2-{130m,1.3b}

[5]Apache License 2.0, https://huggingface.co/spaces/evaluate-metric/{bleu,rouge,meteor}

Model	Mamba										Mamba-2		
Method / Dataset	RTE	MRPC	CoLA	SST-2	QNLI	QQP	MNLI	DART	SAMSum	Spider	DART	SAMSum	Spider
LoRA	2e-03	2e-03	4e-05	2e-03	1e-03	1e-03	2e-03	4e-03	2e-03	4e-03	4e-03	2e-03	4e-03
Additional-scan	4e-03	2e-03	2e-03	1e-01	2e-03	4e-02	4e-03	4e-03	4e-03	4e-03	2e-02	4e-03	1e-02
SDT	1e-03	4e-02	1e-01	4e-02	2e-02	2e-02	1e-01	4e-02	2e-02	4e-02	-	-	-
Initial State Tuning	4e-04	1e-03	2e-03	2e-03	2e-03	2e-03	2e-03	2e-03	2e-04	1e-03	4e-03	2e-04	4e-04
State-offset Tuning ($\boldsymbol{h}$)	1e-03	2e-04	2e-04	1e-04	1e-04	4e-05	4e-04	4e-04	1e-04	2e-04	1e-03	2e-05	2e-05
State-offset Tuning ($\boldsymbol{h}$) (low rank)	-	-	-	-	-	-	-	-	-	-	4e-03	2e-04	2e-04
State-offset Tuning (y)	1e-03	2e-03	1e-03	1e-03	2e-03	1e-03	1e-03	4e-03	1e-03	2e-03	1e-02	2e-04	1e-03

Table 5: Learning rates for each method and dataset. For Mamba and Mamba-2, learning rates for each method and dataset are determined via a small grid search on a dataset subset. The learning rate yielding the best training loss is chosen as the final rate.

Method /Model	**Mamba 130M**	**Mamba 1.4B**	**Mamba-2 130M**	**Mamba-2 1.3B**

LoRA	Rank = 8
	α = 8
	Dropout = 0.1
	Modules = all weight matrices in S6
		Rank = 8
		α = 8
		Dropout = 0.1
		Modules = all weight matrices in S6
			Rank = 16
			α = 16
			Dropout = 0.1
			Modules = all weight matrices in SSD
				Rank = 16
				α = 16
				Dropout = 0.1
				Modules = all weight matrices in SSD

Additional-scan	#States = 8	#States = 8	#States = 32	#States = 32
SDT	Freeze #Channels = 50.0% Freeze #States = 75.0%	Freeze #Channels = 50.0% Freeze #States = 75.0%	-	-
Initial State Tuning	-	-	-	-
State-offset Tuning ($\boldsymbol{h}$)	-	-	-	-
State-offset Tuning ($\boldsymbol{h}$) (low rank)	-	-	Rank = 32	Rank = 64
State-offset Tuning (y)	-	-	-	-

Table 6: Hyperparameter settings for each model and PEFT method. In general, we adjust hyperparameters to maintain a similar number of trainable parameters.

Model	**Method**	**Params** (%)	**Mem.** (GB)	**Latency** (s)
130M	State-offset Tuning ($\boldsymbol{h}$)	0.45	**4.2**	**0.13**
	LoRA	0.35	5.44	0.18
370M	State-offset Tuning ($\boldsymbol{h}$)	0.42	**9.36**	**0.33**
	LoRA	0.32	11.56	0.45
790M	State-offset Tuning ($\boldsymbol{h}$)	0.3	**13.91**	**0.49**
	LoRA	0.23	17.17	0.61
1.4B	State-offset Tuning ($\boldsymbol{h}$)	0.23	**18.77**	**0.67**
	LoRA	0.17	22.99	0.8
2.8B	State-offset Tuning ($\boldsymbol{h}$)	0.19	**31.49**	**1.13**
	LoRA	0.14	37.84	1.33

Table 7: Training speed and memory usage. For each Mamba size, we compare the maximum memory usage and mean latency for processing a single batch during training. Our State-offset Tuning ($\boldsymbol{h}$) is compared against LoRA, as it demonstrated the most similar performance in the experiment section. We configure LoRA to use fewer trainable parameters than State-offset Tuning ($\boldsymbol{h}$). Despite this, State-offset Tuning ($\boldsymbol{h}$) still consumes less memory and is faster in training.

FLOP Overhead While it is possible to avoid extra FLOP with LoRA in constrained single-task settings by merging weights into the pretrained model, real-world serving scenarios often require a single pretrained model to support multiple downstream tasks simultaneously via multiple LoRA adapters. In such cases, avoiding extra FLOP would require storing separately merged models for each task in memory—an inefficient solution. Alternatively, merging weights dynamically at inference time introduces significant computational bottlenecks. As a result, many recent works focus on serving many LoRA adapters efficiently without weight merging (Sheng et al., 2023).

Sequence Length		**L=128**	**L=256**	**L=512**	**L=1024**	**Relative**
Model	**Method**	**GFLOP**				(%)
130M	Pretrained	16.45	32.90	65.81	131.61	100.000
	State-offset Tuning ($\boldsymbol{h}$)	16.46	32.91	65.83	131.65	+ 0.029
	LoRA	16.61	33.21	66.42	132.84	+ 0.937
370M	Pretrained	47.35	94.69	189.39	378.77	100.000
	State-offset Tuning ($\boldsymbol{h}$)	47.36	94.72	189.44	378.87	+ 0.027
	LoRA	47.76	95.52	191.03	382.06	+ 0.867
790M	Pretrained	101.22	202.44	404.88	809.75	100.000
	State-offset Tuning ($\boldsymbol{h}$)	101.24	202.48	404.95	809.90	+ 0.019
	LoRA	101.84	203.67	407.34	814.67	+ 0.608
1.4B	Pretrained	175.23	350.45	700.90	1401.79	100.000
	State-offset Tuning ($\boldsymbol{h}$)	175.25	350.50	701.00	1401.99	+ 0.014
	LoRA	176.05	352.09	704.17	1408.35	+ 0.468
2.8B	Pretrained	353.66	707.32	1414.63	2829.25	100.000
	State-offset Tuning ($\boldsymbol{h}$)	353.70	707.40	1414.80	2829.59	+ 0.012
	LoRA	355.03	710.05	1420.09	2840.17	+ 0.386

Table 8: FLOP overhead across various model sizes and sequence lengths. State-offset Tuning adds less than 0.03% overhead, whereas LoRA incurs over 30× more extra FLOP compared to ours.

Given these practical considerations, we evaluate LoRA without weight merging and conduct experiments comparing the additional FLOP of LoRA and our State-offset Tuning method during inference. We use ptflops (Sovrasov, 2018-2024) to measure computational overhead. As shown in Table 8, our method adds less than 0.03% overhead, while LoRA results in more than 30 times the additional FLOP compared to ours. These results highlight the superior FLOP efficiency of our method compared to LoRA.

Mamba 2.8B Results Table 9 shows the experimental results using Mamba 2.8B. Our State-offset Tuning ($\boldsymbol{h}$) outperforms all methods except full fine-tuning.

Mamba Results on GLUE Dataset Table 10 shows the full results on the GLUE dataset using Mamba 130M. Our State-offset Tuning ($\boldsymbol{h}$) achieves the highest average score among all PEFT methods.

G.2 Mamba-2 Results

Table 11 shows experimental results with Mamba-2 (Dao and Gu, 2024) models. State-offset Tuning ($\boldsymbol{h}$) with low-rank adaptation (Sec. C) significantly reduces the number of trainable parameters. It outperforms existing methods on the Spider benchmark by a large margin and achieves performance comparable to other approaches on the SAMSum and DART datasets.

G.3 State-offset Tuning in SSMs vs. Prefix-Tuning in Transformers

To highlight the effectiveness of State-offset Tuning, we compare its performance with Prefix-Tuning on the Transformer model Pythia (Biderman et al., 2023). We conduct full fine-tuning and Prefix-Tuning experiments on Pythia 160M on GLUE tasks. The results are shown in Table 12.

Full fine-tuning on Mamba 130M generally surpasses Pythia 160M, consistent with Gu and Dao (2024). Prefix-Tuning on both Mamba and Pythia reaches about 85–90% of their full fine-tuning performance.

Our State-offset Tuning achieves approximately 98% of full fine-tuning performance, effectively closing the gap. This success highlights its precise design for SSM-based models.

G.4 Comparison to Selective Dimension Tuning (SDT)

We additionally compare our method with Selective Dimension Tuning (SDT) (Galim et al., 2024), a technique derived from theoretical analysis of SSMs. Note that the hyperparameter selection differs from that used in Galim et al. (2024) to ensure the parameter count is more comparable to ours. As shown in Table 13, our method consistently outperforms SDT in most cases while using fewer parameters.

Model Size		Mamba 2.8B					
Dataset		Params (%)	Spider				
Type	Method		All	Easy	Medium	Hard	Extra
-	Full Fine-tuning (All)	100.00	71.8	87.5	73.5	63.8	51.8
	Full Fine-tuning (S6)	4.44	65.7	81.9	68.8	58.0	41.0
Parameter based	LoRA	0.38	63.9	86.3	**68.2**	49.4	34.3
	BitFit	0.02	59.9	82.3	60.8	**52.9**	31.3
	Additional-scan	0.28	35.0	62.0	31.9	27.4	12.1
Prompt based	Prompt Tuning	0.01	50.7	75.4	53.8	37.4	19.3
	Prefix-Tuning	10.82	45.1	75.0	45.1	32.2	13.9
State based	Initial State Tuning	0.19	59.7	82.3	62.3	43.7	35.5
	State-offset Tuning (h)	0.19	**65.0**	**89.1**	65.9	51.7	**40.4**
	State-offset Tuning (y)	0.01	63.1	85.9	64.1	52.3	37.3

Table 9: Experimental results of fine-tuning the SSM module using pretrained Mamba 2.8B. State-offset Tuning ($\boldsymbol{h}$) stands out as the most effective method among all PEFT approaches.

Model Size		Mamba 130M								
Dataset		Params (%)	GLUE							
Type	Method		RTE	MRPC	CoLA	SST-2	QNLI	QQP	MNLI	Avg.
-	Full Fine-tuning (All)	100.00	71.1	80.6	63.2	92.2	87.4	87.9	80.8	80.5
	Full Fine-tuning (S6)	4.31	69.7	78.9	59.1	91.5	88.1	87.5	80.5	79.3
Parameter based	LoRA	0.92	66.1	78.7	**57.8**	90.8	**87.8**	**86.9**	**79.8**	78.3
	BitFit	0.06	69.5	80.4	54.7	92.0	86.2	85.3	77.2	77.9
	Additional-scan	0.68	57.9	74.0	38.6	79.0	79.9	70.5	36.9	62.4
Prompt based	Prompt Tuning	0.04	56.0	71.6	12.0	89.4	76.8	79.6	61.5	63.8
	Prefix-Tuning	22.69	67.5	75.7	43.4	91.5	83.4	83.1	35.6	68.6
State based	Initial State Tuning	0.45	66.8	78.4	53.0	**92.4**	86.4	86.1	78.5	77.4
	State-offset Tuning (h)	0.45	67.4	**80.8**	56.2	91.9	87.7	85.6	79.7	**78.5**
	State-offset Tuning (y)	0.03	**70.0**	79.6	52.5	91.7	86.3	85.6	78.2	77.7

Table 10: Full results of fine-tuning the SSM module on the GLUE dataset using pretrained Mamba 130M. Our State-offset Tuning ($\boldsymbol{h}$) achieves the highest average score among all PEFT methods.

Model Size		Mamba-2 1.3B									Mamba-2 130M		
Dataset		Params (%)	Spider					SAMSum			Params (%)	DART	
Type	Method		All	Easy	Medium	Hard	Extra	R1	R2	RL		MET.	BLEU
-	Full Fine-tuning (All)	100.00	64.8	85.9	65.7	54.0	42.2	51.0	26.9	42.5	100.00	66.6	34.9
	Full Fine-tuning (SSD)	2.42	55.1	76.2	56.1	42.5	34.3	50.5	26.3	42.4	4.17	65.7	39.7
Parameter based	LoRA	0.37	45.4	69.0	44.4	37.4	21.1	49.7	25.9	41.7	0.76	**70.3**	**49.6**
	BitFit	0.02	50.9	71.4	51.6	45.4	24.1	**50.9**	26.5	**42.6**	0.03	66.2	39.0
	Additional-scan	0.47	31.9	57.3	30.5	23.0	7.2	43.0	20.1	34.8	0.91	58.5	16.0
Prompt based	Prompt Tuning	0.01	45.2	62.5	46.9	34.5	25.9	49.6	26.1	41.6	0.04	65.5	36.9
	Prefix-Tuning	6.99	47.4	71.0	48.2	32.2	25.9	50.8	26.5	**42.6**	12.81	69.2	46.5
State based	Initial State Tuning	1.84	54.3	73.4	57.2	45.4	27.1	50.4	26.4	42.3	3.53	65.3	37.2
	State-offset Tuning (h)	1.84	58.5	**79.3**	61.6	44.6	**33.7**	48.8	24.7	40.5	3.53	70.0	46.3
	State-offset Tuning (h) (low rank)	0.35	**60.5**	79.0	**65.7**	**52.3**	27.7	50.4	**26.8**	42.5	0.72	69.8	47.9
	State-offset Tuning (y)	0.01	43.6	66.5	42.1	36.9	21.1	50.3	26.2	42.2	0.03	65.9	38.7

Table 11: Experimental results of fine-tuning the SSM module using pretrained Mamba-2 (Dao and Gu, 2024) models. We evaluate Spider and its subsets with execution accuracy, SAMSum using ROUGE-1/2/L scores, and DART through METEOR and BLEU scores. State-offset Tuning ($\boldsymbol{h}$) with low-rank adaptation (Sec. C) significantly reduces trainable parameters. It outperforms existing methods on Spider by a wide margin and matches the performance of other approaches on SAMSum and DART.

Dataset		Params	GLUE								
Model	**Method**	(%)	**RTE**	**MRPC**	**CoLA**	**SST-2**	**QNLI**	**QQP**	**MNLI**	**Avg.**	**Relative (%)**
Pythia 160M	Full Fine-tuning	100.00	64.3	77.0	20.5	88.7	85.0	88.8	79.2	71.9	100
	Prefix-Tuning	8.36	57.4	75.0	4.6	88.2	81.5	80.6	62.2	64.2	89
Mamba 130M	Full Fine-tuning	100.00	71.1	80.6	63.2	92.2	87.4	87.9	80.8	80.5	100
	Prefix-Tuning	22.69	67.5	75.7	43.4	91.5	83.4	83.1	35.6	68.6	85
	State-offset Tuning (h)	0.45	67.4	80.8	56.2	91.9	87.7	85.6	79.7	78.5	98

Table 12: Prefix-Tuning experiments on Pythia 160M and Mamba 130M on GLUE tasks. State-offset Tuning for Mamba achieves approximately 98% of full fine-tuning performance, while Prefix-Tuning reaches about 85–90% in both SSM and Transformer architectures.

Model Size	**Mamba 1.4B**										**Mamba 130M**			
Dataset	**Params**	**Spider**					**SAMSum**			**Params**	**DART**		**GLUE**
Method	(%)	All	Easy	Medium	Hard	Extra	R1	R2	RL	(%)	MET.	BLEU	Avg.
SDT	0.26	19.8	38.3	16.6	16.1	4.8	46.3	21.5	37.7	0.51	67.5	**48.2**	63.7
State-offset Tuning (h)	0.23	**57.4**	**77.4**	**59.9**	**44.8**	**33.7**	**50.9**	**26.5**	**42.4**	0.45	**70.0**	47.0	**78.5**
State-offset Tuning (y)	0.01	53.0	**77.4**	55.4	40.8	22.9	50.6	26.1	42.0	0.03	66.8	45.2	77.7

Table 13: Comparison with Selective Dimension Tuning (SDT) (Galim et al., 2024) on Spider, SAMSum, DART, and GLUE. Our method outperforms SDT in most cases while using fewer parameters. Note that the hyperparameter configuration of SDT differs from that in Galim et al. (2024) to ensure a more comparable parameter count.