READ: Recurrent Adaptation of Large Transformers

Sid Wang* **John Nguyen*** **Ke Li** **Carole-Jean Wu**

AI at Meta

Abstract

In the realm of Natural Language Processing (NLP), large-scale transformers have established themselves as pivotal, achieving unparalleled results across numerous tasks. The conventional approach involves pre-training these models on extensive web-scale data, followed by fine-tuning them for specific downstream tasks. However, the burgeoning size of these models, which has surged almost two orders of magnitude faster than GPU memory since 2018, has rendered their fine-tuning financially and computationally exorbitant, limiting this capability to a select few well-funded institutions. Parameter-efficient transfer learning (PETL) has emerged as a potential solution, aiming to efficiently adapt pre-trained model parameters to target tasks using smaller, task-specific models. Nonetheless, existing PETL methods either introduce additional inference latency or marginally reduce memory requirements during training, thus not fully addressing the primary motivation behind PETL. This paper introduces REcurrent ADaption (READ), a novel, lightweight, and memory-efficient fine-tuning method that incorporates a small RNN network alongside the backbone model. READ not only achieves comparable model quality to traditional fine-tuning, saving over 84% in energy consumption, but also demonstrates scalability and independence from the backbone model size. Through extensive experiments on various NLP benchmarks, including the GLUE benchmark, READ showcases robust performance and high efficiency, reducing model training memory consumption by 56% and GPU energy usage by 84% relative to full-tuning, without significantly impacting inference latency and memory. We provide a theoretically justified, scalable solution for fine-tuning large transformers.

1 Introduction

Large-scale transformers architecture have achieved state-of-the-art results in several Natural Language Processing (NLP) tasks [2, 5, 22, 23, 25, 33]. Scaling up the size of these models has been shown to confer various benefits, such as improved model prediction performance and sample efficiency [9, 14, 34]. The conventional paradigm is to pre-train large-scale models on generic web-scale data and fine-tune the models to downstream tasks. However, fine-tuning these models has become prohibitively expensive.

Since 2018, the model size has increased by almost two orders of magnitude faster than GPU memory [20], resulting in prohibitively high cost to advance AI technologies [36]. Only a few well-funded institutions have the resources to fine-tune these models. Parameter-efficient transfer learning (PETL) [1, 13, 15, 16, 18, 19, 38] has emerged as a promising solution to overcome the challenges of full fine-tuning. Parameter-efficient transfer learning techniques aim to address these challenges by leveraging smaller and more task-specific models to efficiently adapt the pre-trained model's parameters to the target task.

*Equal contribution. Correspondence emails: yuwang2020@meta.com, ngjhn@meta.com

37th R0-FoMo: Workshop on Robustness of Few-shot and Zero-shot Learning in Foundation Models at NeurIPS 2023(NeurIPS 2023).

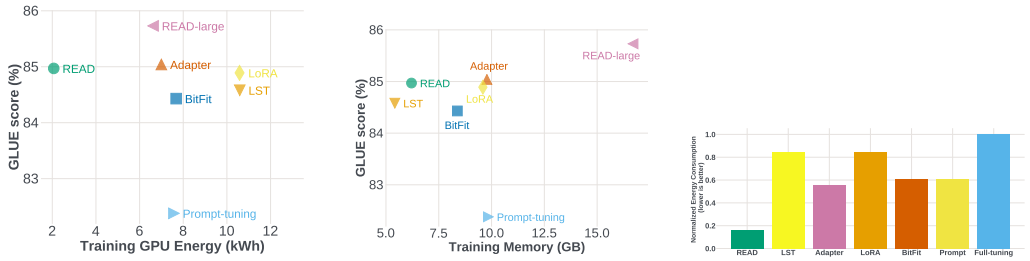

Figure 1: (Left) Comparison of READ and other fine-tuning methods over GLUE tasks on training energy. (Center) Peak training memory relative to full-tuning. (Right) Normalized energy consumption relative to full-tuning on GLUE tasks.

However, all these methods either come with additional inference latency [13] or reduces only a small amount of memory requirement during training — the primary motivation of PETL. Figure 1 illustrates that parameter-efficient methods, while tuning only a small percentage of the overall parameters, still consume significant energy to fine-fine. Since the updated parameters are inside the backbone language models, to calculate gradients for these parameters for backpropagation, PETL methods need to run the backward pass through the sizeable pre-trained language models. This prevents PETL methods from being applied to many real-world applications with limited computational resources. Recent works of Side-Tuning [39] and Ladder-Side Tuning (LST) [29] propose to use a side network that takes intermediate activations from the backbone networks to reduce the need to backpropagate through the large backbone layer. It thus reduces training memory requirement. However, both Side-Tuning and LST have significant drawbacks. In Side-Tuning, the side network only consumes the original inputs, leaving the informative intermediate results from the backbone unused. LST overcomes this problem by using a side Transformer. However, Transformers are challenging to train [21]. Moreover, LST requires an extra pretraining stage to extract a sub-Transformer from the backbone and use it to initialize the side network, increasing the cost of fine-tuning. Additionally, the size of the side-Transformer and the backbone increase, making this approach hard to scale (Figure 4). In this work, we introduce **RE**current **AD**aptation (READ), a novel, lightweight, and efficient fine-tuning method that incorporates a small Recurrent Neural Network (RNN) alongside the backbone model. READ not only achieves comparable model quality to traditional fine-tuning but also realizes more than 84% energy savings during the process.

Contributions

We present READ, a memory and parameter-efficient fine-tuning method that:

1. Introduces a side-tuning design that requires no pretraining of the side network, addressing limitations of previous PETL and side-tuning methods.
2. Demonstrates robust performance and efficiency across various NLP benchmarks, reducing model training memory consumption by 56% and GPU energy usage by 84% relative to full-tuning.
3. Proves scalable for fine-tuning large transformers, independent of the backbone model size.
4. Offers theoretical justification for utilizing the backbone hidden state for side-tuning.

2 Breaking Down REcurrent ADaptation (READ)

2.1 What is READ?

Figure 2 illustrates the READ fine-tuning mechanism on an encoder-decoder transformer backbone, $\mathcal{T}$, which is frozen during training. READ, initialized at both encoder and decoder, primarily consists of a standard RNN and a *Joiner* network, facilitating the amalgamation of multiple information sources to generate RNN inputs. The forward pass through $\mathcal{T}$ is independent of READ, with intermediate results cached at each transformer layer, followed by iterative computation of RNN hidden states and the addition of RNN and $\mathcal{T}$ outputs to derive the final state.

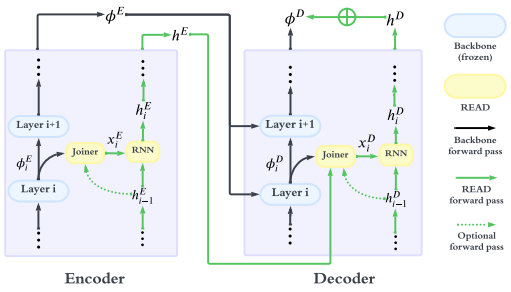

Figure 2: READ Fine-Tuning Mechanism.

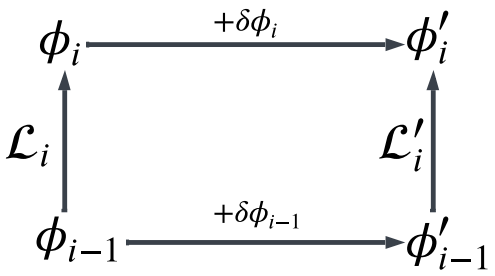

Figure 3: Commuting diagram for *correction*.

Key properties of READ include:

1. **Isolation from Backbone**: Forward pass is separate from $\mathcal{T}$, preventing backward propagation through it and reducing training memory [29].
2. **Simplicity and Efficiency**: Involves only RNNs and FFNs, enhancing usability and training efficiency without requiring pre-training.
3. **Parameter Scalability**: The recurrent nature ensures trainable parameters do not increase with backbone layers, exhibiting sub-linear growth with backbone size.
4. **Unmodified Intermediate Results Consumption**: READ utilizes without altering the backbone's intermediate results[2].

2.2 How does READ work?

Figure 2 provides a visual representation of the READ fine-tuning mechanism, applied to an encoder-decoder transformer backbone, denoted as $\mathcal{T}$. In simpler terms, READ learns the necessary adjustments, or corrections, to the output hidden states at each layer of $\mathcal{T}$ to adapt it for a new task.

Definition 2.1 (*Correction*)**.** If $\mathcal{T}'$ is a modified version of $\mathcal{T}$ and ϕ_i' represents the hidden states at layer $\mathcal{L}_i'$ of $\mathcal{T}'$, the difference $\phi_i' - \phi_i$ is termed a *correction* to ϕ_i, and is denoted as $\delta\phi_i$.

In essence, a correction ($\delta\phi_i$) represents the difference between the hidden states of the original and modified transformer layers, as illustrated in Figure 3. **Key Insights into READ**

Separation from Other Methods: Unlike many fine-tuning methods that directly alter ϕ_i by updating backbone weights or injecting learnable parameters, READ focuses on learning the *correction* needed for a new task.

Practical Application: In practice, we utilize a neural network, named READ, to model the equation system. This involves employing a *Joiner* network to compute x_i, substituting various mathematical entities in the equation with Feed-Forward Networks (FFNs) or linear layers and merging learnable parameters across all indices i for efficiency and scalability. Additionally, Recurrent Neural Networks (RNN) are utilized to model part of the equation system.

This approach does not involve attention mechanisms and operates only on the column space of ϕ, ensuring that all operations are executed efficiently and effectively.

3 Experiment Setup

Baseline and Other State-of-the-Art Designs We compare READ against full tuning and other commonly-used PETL methods. *Full tuning* is not an efficient fine-tuning method but serves as a strong baseline for performance. *BitFit [3]* tunes only bias terms of the model during training. *Prompt-tuning [17]* inserts trainable prompt vectors to the inputs' embedding vectors. *Adapters [13]*

[2]Notably, although not detailed in this paper, READ enables *multi-tasking* with multiple networks, necessitating only a single backbone pass, thereby reducing training and inference costs.

appends a small residual MLP after every attention and feed-forward layer. We experiment with the sequential adapter version by Houlsby et al. [13]. *LoRA [15]* inserts trainable low-rank matrices into each layer of the backbone Transformer model to parameterize the weights' changes. *LST [29]* hierarchically adds multiple side networks, with each side network responsible for modulating the activations of a specific layer in the pre-trained model. For all PETL methods and READ, we keep the backbone transformer frozen throughout the training and only update the new parameters.

Datasets. We evaluate READ and the baselines on the GLUE [31] benchmarks. These benchmarks cover a variety of NLP tasks, including linguistic acceptability (CoLA [32]), paraphrase detection (MRPC [10], QQP [8], STS-B [6]), natural language inference (MNLI [35], QNLI [27]), and sentiment classification (SST-2)[3]. In GLUE, the original test set labels are not publicly available. Instead, we follow [40] and [16] to create a test set for each task as follows: if the training set contains less than 10k samples, we equally split the original validation set into two subsets and treat them as new validation and test sets; otherwise, we use the original validation set as the test set, and split 1k from the training set as the new validation set. For MNLI, we use the mismatched set as the validation set and the matched set as the test set. We report the dataset sizes in Appendix D.1.

Model Specification and Experimental Details. We adopt the encoder-decoder T5 [24] model as our backbone transformer. We use T5_{BASE} for all of our experiments, and also use T5_{LARGE} for READ experiments, which we denote by READ-large. We perform fine-tuning on each dataset for up to 30 epochs and do an early stop once validation metric stops improving. For READ, we experiment with $\{128, 256\}$ as RNN hidden dimensions, $\{$RNN, LSTM, GRU$\}$ as RNN architectures. For PETL baselines, we experiment with $\{32, 64\}$ as Adapters' bottleneck sizes, $\{8, 32\}$ as LoRA's ranks, and $\{10, 20, 30\}$ as Prompt-tuning's prompt sizes. For all experiments, we conduct a grid search for learning rates in between $[1 \times 10^{-6}, 3 \times 10^{-3}]$ on log scale for up to 32 rounds. We choose the hyperparameters that achieve the best validation scores and report their test scores. Complete setup and hyperparameters detail are in Appendix D.2.

4 Evaluation Results

We train and evaluate each method on all the GLUE tasks. We take the cumulative energy consumption and measure the peak GPU during training. In this section, we report and analyze the results on the GLUE Benchmarks.

READ outperforms other methods while consuming significantly lower energy: Figure 1 (left) shows that READ can reduce GPU energy consumption by up to 90% compared to full-tuning. READ lowers the GPU memory footprint by 56% while retaining the same model accuracy when retraining. While other parameter-efficient transfer learning (PETL) methods like LoRA, BitFit or Adapter reduce the number of trainable parameters, they do not reduce the compute cost required to fine-tune. We believe the underlying optimization objective for PETL is to reduce this compute cost. Table 1 shows the performance of all methods on GLUE with T5BASE. Excluding Adapter, READ outperforms all parameter-efficient methods while consuming at least 68% less energy. Compared to Adapter, READ achieves nearly the same model accuracy (less than 0.1% lower) while using 70% less energy. More interestingly, READ with T5LARGE (i.e. READ-large) achieves better performance than all other methods and consumes similar or less energy compared to other methods. For example, READ-large outperforms Full-tuning and Adapter by 1.4% and 0.8% with 69% and 5% less energy, respectively. These results show that by using READ, we can scale up the model size while keeping the same hardware and memory constraints.

READ consumes less training memory: Figure 1 (right) shows the design space trade-off between model quality performance and memory footprint. READ improves the training memory requirement by at least 25% compared to all the other baselines while achieving similar or better performance. READ with T5_{LARGE} consumes similar amount of memory as full-tuning with T5_{BASE}. As the backbone size increases, the memory savings achieved by READ become increasingly significant in comparison to the other PETL methods, as depicted in Figure 4 (right). Notably, at the T5_{3B} backbone level, these savings reach as high as 43%. This observation suggests that READ is remarkably effective in the regime of fine tuning large Transformers.

[3]We exclude RTE from GLUE due to its small size compared to other tasks

Table 1: Performance and energy consumption results of all methods on GLUE tasks. We report the accuracy for SST-2, MNLI, QNLI, and Matthew's Correlation for CoLA. For STS-B we report the average of Pearson correlation and Spearman correlation. For MRPC and QQP, we report the average of F1 score and accuracy. For all tasks, we report the average score on 3 different seeds. Bold fonts indicate the best results of that column.

Method	Trainable Params (%)	Power (kW)	Energy (kWh)	CoLA	MNLI	QNLI	MRPC	QQP	SST-2	STS-B	Avg.
Baselines											
Full-tuning	100	0.77	12.52	53.97	86.17	90.87	86.88	89.71	92.89	88.19	84.52
Adapter	0.96	0.50	6.99	52.56	85.68	92.89	87.84	88.95	93.12	87.51	85.04
LoRA	0.48	0.68	10.58	51.70	85.20	92.72	**88.07**	88.92	93.46	86.97	84.89
BitFit	0.06	0.47	7.68	50.92	85.28	92.58	86.32	88.70	**94.15**	86.94	84.43
Prompt-tuning	**0.01**	0.50	6.45	42.71	79.38	91.73	86.04	88.74	93.12	84.96	82.38
LST	2.00	0.44	10.59	53.38	84.53	92.43	87.38	88.31	92.09	87.37	84.58
Our method											
READ	0.80	**0.43**	**2.06**	52.59	85.25	92.93	87.09	89.10	93.80	87.77	84.97
READ-large	0.32	0.62	6.62	**54.05**	**87.29**	**93.68**	87.70	89.34	**93.92**	**88.58**	**85.73**

Table 2: READ with and without recurrency

Method	CoLA	MNLI	QNLI	MRPC	QQP	SST-2	STS-B	Avg.	Trainable Params (%)
READ (with Recurrency)	52.59	85.25	92.93	87.09	89.10	93.80	87.77	84.97	0.8
READ (w/o. Recurrency)	53.24	84.10	91.51	89.02	89.18	94.04	87.10	85.15	9.6
READ-large (with Recurrency)	54.05	87.29	93.68	87.70	89.34	93.92	88.58	85.73	0.32
READ-large (w/o Recurrency)	50.17	86.90	93.00	87.61	89.12	94.15	87.46	85.02	6.4

READ is scalable: As shown in Figure 4 (left), the number of trainable parameters of READ scale more slowly as compared to the other PETL methods. READ's number of parameters exhibits a log-linear growth pattern as the T5 backbone model size increases. In fact, the recurrent nature of READ makes its tunable size independent from the number of backbone layers, making READ a more suitable choice for fine-tuning large Transformers in practice.

The importance of recurrency We perform ablation analysis on the importance of recurrence in READ in Table 2. We find that the removal of recurrence does not significantly enhance READ quality and even diminishes quality for the T5 large backbone. However, without recurrence leads to over 12 times more trainable parameters, compromising scalability.

Comparison with Ladder-Side-Tuning (LST) We compare our methods with Ladder-Side-Tuning (LST), another memory efficient fine-tuning approach [29]. We follow the pruning method introduced in [29] to extract a smaller transformer from the backbone transformer and use it to initialize the side transformer, and re-implement LST. Table 1 lists the results of LST (using T5_{BASE}) on GLUE benchmarks and its energy efficiency.The results indicate that READ (base) outperforms LST (base) on most tasks (except for a tiny task MRPC), using $\mathbf{80\%}$ **less energy consumption** and $\mathbf{60\%}$ **less trainable parameters**. While LST consumes 15% less peak training memory relative to READ, it takes $\mathbf{40\%}$ **more inference memory** and $\mathbf{77\%}$ **longer inference time** than READ, a consequence of its attention-based side-network architecture. It is also noteworthy that when compared to LST even READ-large saves 38% GPU energy and yields a similar inference latency, with 1.4% relative gain on the averaged GLUE score. Furthermore, the "pre-training stage" refers to the process described in LST paper section 2.2, where distillation is performed with T5 pre-training objective. It is important to note that caching the attention outputs does not involve updating any model parameters and should not be considered as a form of training.

5 Related Work

In this section, we summarize the closely related works to ours and leave the more detailed discussion to Appendix C.

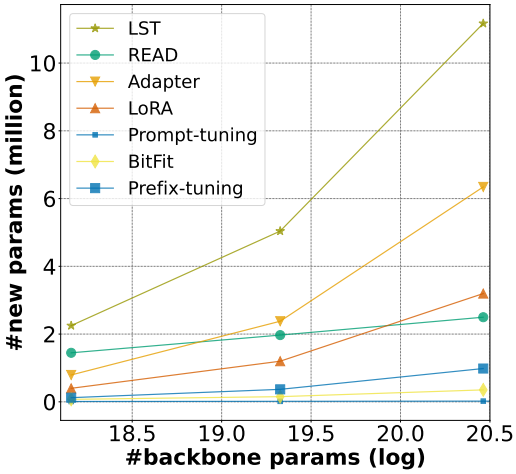

Figure 4: The number of trainable parameters as the backbone model size increases.

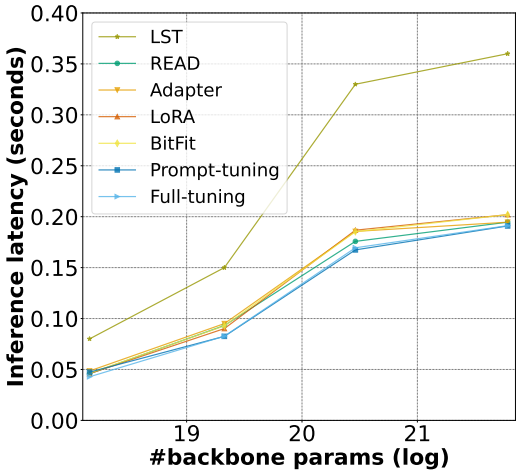

Figure 5: Inference latency as backbone model size increases.

Parameter-efficient Transfer Learning. The surge in generative AI applications [4, 28, 30, 33] has been hindered by the computational and memory costs of fine-tuning large transformers. Parameter-efficient transfer learning (PETL) [1, 13, 18–20, 29, 38] addresses this by training a minimal parameter set, with various methods like Adapter-based approaches, Low-Rank Adaptation (LoRA), BitFit, and Prompt-tuning. Unlike these, READ introduces memory efficiency by incorporating a small recurrent network into the backbone, focusing on reducing memory usage over parameter minimization.

Memory-Efficient Training. Memory-efficient training strategies, such as gradient checkpointing [7], reversible layers [11], ZeRO [26], and Layer-wise Adaptive Rate Scaling (LARS) [37], aim to mitigate memory consumption by optimizing the storage and computation of intermediate activations and model states, particularly in distributed training environments.

Sidenet Tuning. Side-tuning [39] and Ladder side-tuning [29] employ lightweight side networks to adapt pre-trained model activations for new tasks without modifying the base model. READ, while inspired by these, distinguishes itself by utilizing a single RNN block that processes the backbone network's hidden state recurrently, ensuring the fine-tuning parameter count does not scale with the backbone size. Unlike Side-Tuning, READ iteratively calculates its RNN states across all layers and exclusively employs RNN and Feed-Forward Network (FNN) structures, negating the need for transformers or attention mechanisms and enabling use without pre-training.

6 Conclusion and Limitations

Limitations. Due to our limited computing resources, we could not scale the backbone to an even larger scale. A future direction is to fine-tune READ on Llama-7B [30] or even larger variants. Another direction can be studied if READ can generalize well in a low-data regime. A drawback of READ is its tendency to require more epochs to converge on small datasets than other PETL methods. Consequently, although READ is more efficient in per-unit time computations, it may not yield significant overall consumption gains when a task has few data points. We leave investigating READ on the low-data regime as future work.

Conclusion. In this paper, we propose REcurrent ADaption (READ), a lightweight and efficient parameter and memory-efficient fine-tuning method, for large-scale transformers. We show that READ achieves comparable accuracy to full fine-tuning while saving more than 84% of energy consumption and reducing training memory consumption by 56% relative to full-tuning. We demonstrate the scalability of READ because READ is independent of backbone model size. We hope that READ can make fine-tuning large models more accessible to a broader range of researchers and applications.

References

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

[2] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021.

[3] Elad Ben-Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *ArXiv*, abs/2106.10199, 2021.

[4] Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. *arXiv preprint arXiv:2304.01373*, 2023.

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

[6] Daniel Matthew Cer, Mona T. Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *International Workshop on Semantic Evaluation*, 2017.

[7] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost, 2016.

[8] Zihang Chen, Hongbo Zhang, Xiaoji Zhang, and Leqi Zhao. Quora question pairs. 2017.

[9] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

[10] William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *International Joint Conference on Natural Language Processing*, 2005.

[11] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. The reversible residual network: Backpropagation without storing activations, 2017.

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

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

[14] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[15] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[16] Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035, 2021.

[17] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *ArXiv*, abs/2104.08691, 2021.

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

[19] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353.

[20] Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. Scaling down to scale up: A guide to parameter-efficient fine-tuning, 2023.

[21] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the difficulty of training transformers. In *Conference on Empirical Methods in Natural Language Processing*, 2020.

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

[23] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. *Advances in neural information processing systems*, 32, 2019.

[24] Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *ArXiv*, abs/1910.10683, 2019.

[25] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[26] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models, 2020.

[27] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *ArXiv*, abs/1606.05250, 2016.

[28] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[29] Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. *arXiv preprint arXiv:2206.06522*, 2022.

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

[31] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *ArXiv*, abs/1804.07461, 2018.

[32] Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2018.

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

[34] Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, et al. Larger language models do in-context learning differently. *arXiv preprint arXiv:2303.03846*, 2023.

[35] Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *North American Chapter of the Association for Computational Linguistics*, 2017.

[36] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, Michael Gschwind, Anurag Gupta, Myle Ott, Anastasia Melnikov, Salvatore Candido, David Brooks, Geeta Chauhan, Benjamin Lee, Hsien-Hsin Lee, Bugra Akyildiz, Maximilian Balandat, Joe Spisak, Ravi Jain, Mike Rabbat, and Kim Hazelwood. Sustainable ai: Environmental implications, challenges and opportunities. In D. Marculescu, Y. Chi, and C. Wu, editors, *Proceedings of Machine Learning and Systems*, volume 4, pages 795–813, 2022.

[37] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks, 2017.

[38] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models, 2022.

[39] Jeffrey O Zhang, Alexander Sax, Amir Zamir, Leonidas Guibas, and Jitendra Malik. Side-tuning: a baseline for network adaptation via additive side networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 698–714. Springer, 2020.

[40] Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q. Weinberger, and Yoav Artzi. Revisiting few-sample bert fine-tuning. *ArXiv*, abs/2006.05987, 2020.

A Appendix

A.1 Revisit Transformer

In this subsection, we briefly revisit the computation of transformer and introduce some convenient notations for the future. Let $\mathcal{T}$ be a transformer of dimension d with N layers $\mathcal{L}_1, \cdots, \mathcal{L}_N$. At each layer $\mathcal{L}_i$, let the feed-forward network be $\mathcal{F}_i$ and multihead-attention be $\mathcal{A}_i$. Given a context sequence of m tokens, we can express each layer as a mapping from $\mathbb{R}^{m\times d}$ to $\mathbb{R}^{m\times d}$ as follows:

$$\mathcal{L}_i = (\mathcal{F}_i^* + I) \circ \mathcal{A}_i^* + I, \tag{1}$$

where I represents the identity mapping, $\circ$ denotes mapping composition, and $\mathcal{F}_i^* =: \mathcal{F}_i \circ \text{LN}$, $\mathcal{A}_i^* =: \mathcal{A}_i \circ \text{LN}$ (i.e. compositions with layer-normalization). Further, we define $\mathcal{R}_i = (\mathcal{F}_i^* + I) \circ \mathcal{A}_i^*$ so as to write layer mapping as $\mathcal{L}_i = \mathcal{R}_i + I$.

A.2 Derivations of READ

Following the notations in Subsection A.1, we derive an inductive formula for the *corrections*:

$$\begin{aligned}
\delta\phi_i &= \phi_i' - \phi_i \\
&= (\mathcal{R}_i' + I)(\phi_{i-1}') - (\mathcal{R}_i + I)(\phi_{i-1}) \\
&= \mathcal{R}_i'(\phi_{i-1}') - \mathcal{R}_i(\phi_{i-1}) + (\phi_{i-1}' - \phi_{i-1}) \\
&= \mathcal{R}_i'(\phi_{i-1}') - \mathcal{R}_i(\phi_{i-1}) + \delta\phi_{i-1} \\
&= \big(\mathcal{R}_{i-1}' - \mathcal{R}_i\big)(\phi_{i-1}') + \big(\mathcal{R}_i(\phi_{i-1}') - \mathcal{R}_i(\phi_{i-1})\big) + \delta\phi_{i-1} \\
&= \delta\mathcal{R}_i(\phi_{i-1}') + J\mathcal{R}_i\delta\phi_{i-1} + \delta\phi_{i-1}.
\end{aligned} \tag{2}$$

Here $\delta\mathcal{R}_i$ denotes the operator difference $\mathcal{R}_i' - \mathcal{R}_i$, and $J\mathcal{R}_i$ is the Jacobian matrix of $\mathcal{R}_i(\cdot)$ evaluated at some point lying on the line segment from ϕ_{i-1} to ϕ_{i-1}'. To simplify our arguments, we (1) assume that $J\mathcal{R}_i$ takes value at ϕ_{i-1}, (2) let $\mathcal{T}'$ be the consequence of fine-tuning with Adapter or LoRA (applied at FFN layers $\mathcal{F}_i$) [4]. We use $\mathcal{P}$ to denote a common module adopted by Adapter and LoRA which consists of a down projection matrix to a lower dimension possibly followed by a non-linear activation, and then composed with an upper projection back to the original dimension[5]. Under these assumptions, the first term of the RHS in (2) now becomes

$$\begin{aligned}
\delta\mathcal{R}_i(\phi_{i-1}') &= \begin{cases} \mathcal{P}_i \circ (\mathcal{P}_i + I)^{-1}\phi_i' & \text{(Adapter)} \\ \mathcal{P}_i \circ (\mathcal{P}_i + \mathcal{F}_i)^{-1}\phi_i' & \text{(LoRA)} \end{cases} \\
&=: \mathcal{W}_i(\phi_i + \delta\phi_i)
\end{aligned} \tag{3}$$

Now plugging (3) back to (2), we obtain

$$\delta\phi_i = \mathcal{W}_i(\phi_i + \delta\phi_i) + J\mathcal{R}_i\delta\phi_{i-1} + \delta\phi_{i-1}. \tag{4}$$

Notice that both sides of equation (4) contains $\delta\phi_i$. Because of the non-linearity of W_i, there is no straightforward way to extract an inductive formula of $\delta\phi_i$ from (4).

However, let us rewrite equation (4) as

$$\begin{aligned}
&\delta\phi_i - \mathcal{W}_i(\phi_i + \delta\phi_i) - (J\mathcal{R}_i\delta\phi_{i-1} + \delta\phi_{i-1}) \\
=&F(\delta\phi_i, \phi_i, J\mathcal{R}_i\delta\phi_{i-1} + \delta\phi_{i-1}) = 0,
\end{aligned} \tag{5}$$

and compute the Jacobian to see that $J_{\delta\phi_i}F = I - J\mathcal{W}_i$, which is invertible when $\mathcal{P}_i$ (and hence $\mathcal{W}_i$ 3) has small norm. Now by Implicit Function Theorem there exists G such that

$$\delta\phi_i = G(\phi_i, J\mathcal{R}_i\delta\phi_{i-1} + \delta\phi_{i-1}). \tag{6}$$

An alternative argument is to use a first order approximation of $\mathcal{W}_i(\phi_i + \delta\phi_i)$ assuming that $\delta\phi_i$ is sufficiently small, which gives us the following inductive formula:

$$\delta\phi_i = (I - J\mathcal{W}_i)^{-1} \circ \left(\mathcal{W}_i\phi_i + J\mathcal{R}_i\delta\phi_{i-1} + \delta\phi_{i-1}\right) \tag{7}$$

[4] For fine-tuning methods that modify attention, we expect a similar conclusion that demands a more intricate line of reasoning, which we defer to future research.

[5] The operator norm of $\mathcal{P}$ is small when its two matrices have small weights, and therefore addition with $\mathcal{P}$ will not change the invertibility of an already invertible operator.

We take the second approach above and adopt formula (7) as we move forward, because of its explicit function form. Note that every operation in (7) acts on the column space of ϕ except for the Jacobian transform $J\mathcal{R}_i$, so let us first focus on expanding $J\mathcal{R}_i\delta\phi_{i-1}$. In fact, we will compute the Jacobian for a general attention mapping that takes 3 arguments ϕ_q, ϕ_k, ϕ_v (i.e. hidden states of queries, keys, and values), and then apply the results to the special case of self-attention (as in encoder) and cross-attention (as in decoder) respectively. For the sake of brevity, we assume that the number of attention head is 1 and omit the output projection, as neither of which is essential to our conclusion.

Let ϕ_q, ϕ_k, ϕ_v be matrices in $\mathbb{R}^{m_q\times d}, \mathbb{R}^{m_k\times d}, \mathbb{R}^{m_k\times d}$, which stand for $\mathbb{R}^d$-vector representations of the query, key, and value token sequences with length m_q, m_k, m_k respectively. We use an upper index α to denote the vector associated to the α^{th} token, and omit the lower layer index i when no ambiguity is present (e.g. $\mathcal{A}^\alpha$ is the α^{th} column of $\mathcal{A}$'s output.). First, we have

$$J\mathcal{R}_i\delta\phi_{i-1} = (J\mathcal{F}^* + I)\circ J\mathcal{A}\circ J\text{LN}(\delta\phi_{i-1}) \tag{8}$$

by chain rule. Next we expand $J\mathcal{A}$, as every other operation in (8) acts on the column space of ϕ; especially, up to composing with a feed-forward neural network, let us replace JLN by an identity to simplify our notations. A straightforward computation gives the following:

$$\begin{aligned}
J_{\phi_q}\mathcal{A}^\alpha(\delta\phi_q) &= \left[\sum_{\beta=1}^{m_q}\sigma^{\alpha\beta}\cdot\frac{(v^\beta-\mathcal{A}^\alpha)\cdot {k^\beta}^T}{\sqrt{d}}\cdot W_Q\right]\delta\phi_q^\alpha,\\
J_{\phi_k}\mathcal{A}^\alpha(\delta\phi_k) &= \sum_{\beta=1}^{m_k}\sigma^{\alpha\beta}\cdot\frac{(v^\beta-\mathcal{A}^\alpha)\cdot {q^\alpha}^T}{\sqrt{d}}\cdot W_K\delta\phi_k^\beta,\\
J_{\phi_v}\mathcal{A}^\alpha(\delta\phi_v) &= \sum_{\beta=1}^{m_k}\sigma^{\alpha\beta}\cdot W_V\delta\phi_v^\beta.
\end{aligned} \tag{9}$$

Here W_Q, W_K, W_V denote the query, key, and value projection matrices of $\mathcal{A}$, $q^\alpha = W_Q\phi_q^\alpha$, $k^\alpha = W_K\phi_k^\alpha$, and $\sigma^{\alpha\beta} = \text{softmax}({q^\alpha}^T\cdot k^\beta/\sqrt{d})$.

Case 1, $\mathcal{A}$ is self-attention Upon setting ϕ_q, ϕ_k, ϕ_v to ϕ, and $\delta\phi_q, \delta\phi_k, \delta\phi_v$ to $\delta\phi$ in (9), we obtain:

$$\begin{aligned}
J\mathcal{A}^\alpha\delta\phi = &\left[\sum_{\beta=1}^{m_q}\sigma^{\alpha\beta}\cdot\frac{(v^\beta-\mathcal{A}^\alpha)\cdot {k^\beta}^T}{\sqrt{d}}\cdot W_Q\right]\delta\phi^\alpha\\
&+\sum_{\beta=1}^{m_q}\sigma^{\alpha\beta}\cdot\left[\frac{(v^\beta-\mathcal{A}^\alpha)\cdot {q^\alpha}^T}{\sqrt{d}}\cdot W_K + W_V\right]\delta\phi^\beta.
\end{aligned} \tag{10}$$

Note the two quantities in the square brackets are $\mathbb{R}^{d\times d}$-matrix-valued linear functions of values that can be computed from the cached results at $\mathcal{L}_i$, which we shall denote by Φ, Ψ from now on:

$$J\mathcal{A}^\alpha\delta\phi = \Phi\cdot\delta\phi^\alpha + \sum_{\beta=1}^{m_q}\sigma^{\alpha\beta}\Psi\cdot\delta\phi^\beta. \tag{11}$$

Now, upon inserting (11) to the α^{th}-column of (8) by setting ϕ as ϕ_i, and then plugging (8) back in (7), we obtain the iterative formula for outputs h_i:

$$\begin{cases}
\psi_i^\alpha = \Phi_i\cdot\mathcal{F}_i\delta\phi_{i-1}^\alpha + \sum_{\beta=1}^m\sigma_i^{\alpha\beta}\Psi_i\cdot\mathcal{F}_i\delta\phi_{i-1}^\beta\\
x_i^\alpha = [{\phi_i^\alpha}^T, {\psi_i^\alpha}^T]^T\\
\delta\phi_i^\alpha = \mathcal{G}_i(\mathcal{H}_i x_i^\alpha + \delta\phi_{i-1}^\alpha)
\end{cases} \tag{12}$$

where Φ_i, Ψ_i are defined as in (11), and $\mathcal{F}_i, \mathcal{G}_i, \mathcal{H}_i$ are FNNs that simulate JLN, $(I - J\mathcal{W}_i)^{-1}$, and $[\mathcal{W}_i, J\mathcal{F}_i^* + I]$ respectively; see (7) and (8). Note (12) is exactly (**??**) upon replacing $\delta\phi$ by h.

Case 2, $\mathcal{A}$ is cross-attention Since the decoder's *correction* iterative formula follows from a similar line of reasoning as self attention, we present the final results while omitting the details:

$$\begin{cases}
\psi_i^\alpha = \Phi_i\cdot\mathcal{F}_i^D\delta\phi_{i-1}^{D,\alpha} + \sum_{\beta=1}^m\sigma_i^{\alpha\beta}\Psi_i\cdot\mathcal{F}_i^E\delta\phi^{E,\beta}\\
x_i^\alpha = [{\phi_i^\alpha}^T, {\psi_i^\alpha}^T]^T\\
\delta\phi_i^{D,\alpha} = \mathcal{G}_i(\mathcal{H}_i x_i^\alpha + \delta\phi_{i-1}^{D,\alpha})
\end{cases} \tag{13}$$

Table 3: Efficiency results of LST, READ, and Full-tuning. We report the training GPU energy usage summed over all tasks, and the peak training memory (per batch) averaged over all tasks. For inference memory/time, we use MNLI and report the average per batch (with test batch size 1).

	Training GPU Energy (kWh)	Training Memory (GB)	Trainable Params (Million/%)	Inference Time (s)	Inference Memory (GB)
Full-tuning	12.52	17.86	247.58/100.00	**0.083**	**0.943**
LST	10.59	**5.77**	5.04/2.00	0.165	1.358
READ	**2.07**	6.90	**1.97/0.80**	0.093	0.966
READ-large	6.62	17.74	11.17/1.4	0.175	2.943

where an upper index $D\backslash E$ are used to distinguish between the hidden states of decoder and encoder, and $\delta\phi^E$ is the final *correction* of encoder.

A.3 Ablation Experiments

A.4 GPU Energy Analysis

To provide a comprehensive understanding, we include an analysis below to show the mean and standard deviation for the sums of GPU energy, epochs to convergence, and training time across all GLUE tasks. While this analysis does reveal some variations in energy/time levels, they are not significantly substantial to alter the general trend, as READ continues to stand out as the most energy-efficient approach, with faster convergence than most baselines except for full-tuning.

	Full	Adapter	LoRA	Prompt	BitFit	LST	READ
Energy	$12.52_{0.44}$	$6.99_{0.62}$	$10.58_{1.9}$	$6.45_{0.24}$	$7.68_{0.06}$	$10.59_{0.3}$	$2.06_{0.18}$
Epoch	7.0_{0}	$13.01_{0.58}$	$25.84_{3.9}$	$34.85_{2.4}$	$23.61_{1.3}$	$23.58_{1.5}$	$12.31_{1.4}$
Time	$472.74_{12.77}$	$485.2_{51.12}$	$409.4_{19.52}$	$315.53_{5.01}$	$292.89_{4.03}$	$984.55_{19.07}$	$155.11_{8.4}$

Table 4: GPU Energy Consumption and Training Time across 3 trials.

A.5 READ's Inference and Memory Efficiency

As Figure 5 (left) and Table 5 indicate, READ achieves comparable inference latency and memory requirement as the other PETL methods. To assess the inference memory impact of READ more comprehensively, we use Figure 5 (right) to demonstrate that, as the backbone size increases, the inference memory growth (relative to Full-tuning) of READ becomes less noticeable and decays to a similar extent as the other methods at $T5_{LARGE}$.

Table 5: Average inference memory consumption (GB) for every method with different backbones on GLUE benchmark.

	READ	LST	Adapter	LoRA	Prompt	Bias	Full
$T5_{SMALL}$	0.317	0.427	0.303	0.302	0.301	0.301	0.300
$T5_{BASE}$	0.966	1.358	0.952	0.948	0.948	0.945	0.943
$T5_{LARGE}$	2.943	4.597	2.936	2.925	2.925	2.914	2.912
$T5_{3B}$	10.885	11.400	10.878	10.866	10.894	10.855	10.853

Table 6: Split sizes, training GPU number, and training batch size per GPU node for all GLUE tasks.

	CoLA	MNLI	QNLI	MRPC	QQP	SST-2	STS-B
Training Samples (k)	8.5	392.7	99.3	3.7	323.4	66.5	5.8
Test Samples (k)	0.52	9.8	5.4	0.2	40.4	0.9	0.8
Validation Samples (k)	0.52	9.8	1.0	0.2	1.0	1.0	0.8
GPUs	2	8	8	1	8	8	1
Batch Size per GPU	48	12	12	96	12	12	96

B Architecture

B.1 Architecture choices

The matrix functions Ψ, Φ in equation (12) and (13) requires computing dot products for m^2 pairs of vectors (9) with time complexity as large as $O(m^2d^2)$. To reduce latency cost in practice, we make substantial reductions to the first equation in both (12) and (13) for READ experiments in this paper, as listed below:

- Indices is are removed and learnable parameters are fused across all layers;
- In self-attention, we set Ψ, Φ to be constantly zeros; in other words, only hidden states are cached and used for encoder *corrections*;
- In cross-attention, we set Φ to zero and $\Psi \cdot \mathcal{F}_i^E h_{i-1}^\beta =: Lh_{i-1}^\beta$, where L is a learnable linear projection, so besides decoder hidden states we also need to cache the cross-attention scores for computing decoder *corrections*. Furthermore, we use a simple addition operation to combine ϕ_i and ψ_i in (13) instead of a learnable layer.

Note some reductions we made above might be over-simplified but this paper does not explore other more sophisticated[6] while still computationally efficient options, such as a gated neural network:

$$\begin{cases} \Phi \cdot \mathcal{F}_i(h_{i-1}^\alpha) = \text{Gate}(\phi_i^\alpha) \odot \text{FFN}(h_{i-1}^\alpha), \\ \Psi \cdot \mathcal{F}_i(h_{i-1}^\beta) = \text{Gate}(\phi_i^\alpha) \odot \text{FFN}(h_{i-1}^\beta), \end{cases} \tag{14}$$

where $v \odot X = \text{diag}(v) \cdot X$. We leave the pertinent explorations to future works.

B.2 READ Algorithm

Algorithm 1 outlines a forward pass during READ fine-tuning. Let $\mathcal{T}$ be a transformer with N^E encoder layers and N^D decoder layers, and $X \backslash Y$ be source\target sequences of length $m \backslash n$:

C Related Works

Parameter-efficient Transfer Learning. There has been an explosion of generative AI applications in recent months [4, 28, 30, 33]. However, the ability to fine-tune large transformers is primarily limited by the growing compute cost required to adapt and serve these models. Parameter-efficient transfer learning (PETL) [1, 13, 18–20, 29, 38] aims to solve this problem by training only a small set of parameters. There are many PETL methods which we defer the reader to [20] for a more comprehensive overview. In this section, we will summarize the most popular PETL methods which

[6] A more sophisticated choice potentially introduces more dependency on cached results and likely to improve performance at a trade-off of higher number of computation flops.

Algorithm 1 READ Fine Tuning Algorithm

Initialize RNNs $\mathcal{N}^E, \mathcal{N}^D$ and a learnable projection Ψ.

$\{\phi_i^{E,\alpha}\}_{i=1,\alpha=1}^{N^E,m}, \{\phi_j^{D,\alpha}\}_{j=1,\alpha=1}^{N^D,n}, \{\sigma_i^{E,\alpha\beta}\}_{i=1,\alpha=1,\beta=1}^{N^E,m,m}, \{\sigma_j^{D,\alpha\beta}\}_{j=1,\alpha=1,\beta=1}^{N^D,n,m} \leftarrow \mathcal{T}(X,Y)$

$h_{E,0} \leftarrow 0$ ▷ We assume embeddings need no *corrections*.

for i in $1, \cdots, N^E$ **do** ▷ Iteratively compute encoder *corrections*.

$h_i^{E,\alpha} = \mathcal{N}^E(\phi_i^\alpha, h_{i-1}^{E,\alpha}), \forall\alpha$

$h_{D,0} \leftarrow 0$

for j in $1, \cdots, N_D$ **do** ▷ Iteratively compute decoder *corrections*.

$\psi_j^\alpha = \sum_{\beta=1}^m \sigma_{D,j}^{\alpha\beta} \Psi h_{N^E}^{E,\beta}, \forall\alpha$

$x_j^\alpha = \phi_j^\alpha + \psi_j^\alpha, \forall\alpha$

$h_j^{D,\alpha} = \mathcal{N}^D(x_j^\alpha, h_{j-1}^{D,\alpha}), \forall\alpha$

${\phi_{N_D}^D}' \leftarrow \phi_{N^D}^D + h_{N^D}^D$ ▷ Obtain *adapted* outputs.

we used as baselines. Adapter-based approaches [12, 13] insert small learnable modules between pre-trained model layers and only update these adapters during fine-tuning, reducing computational cost and memory requirements. Low-Rank Adaptation (LoRA) [15] injects trainable rank decomposition matrices into each layer of the Transformer model. BitFit [38] fine-tunes only the biases of the model. Prompt-tuning [18] is a successor of Prefix-Tuning [19], which adds a continuous task-specific prompt to the input. In contrast, current PETL approaches aim to minimize the number of parameters trained. These approaches do not lead to memory efficiency, a more meaningful objective than parameter efficiency. This work proposes READ, simple memory-efficient methods by inserting a small recurrent network into the backbone.

Memory-Efficient Training. Memory-efficient training reduces memory consumption by reducing the storage of intermediate activations [29]. Gradient checkpointing [7] reduces memory consumption during backpropagation by storing a subset of intermediate activations and recomputing them as needed, trading time for memory. Reversible layers [11] reconstruct each layer's activations from the next layer's activations. ZeRO [26] partitions model states, gradients, and optimizer states across multiple devices for distributed training, significantly reducing memory redundancy. Layer-wise Adaptive Rate Scaling (LARS) [37] dynamically scales learning rates for different layers, reducing memory overhead associated with large gradients and enabling the training of large models with limited memory.

Sidenet Tuning. Side-tuning [39] adds a lightweight side network alongside the pre-trained model. During training, the side network and the task-specific head are updated while the pre-trained model's parameters are kept fixed. The side network learns to modulate the pre-trained model's activations, allowing it to adapt to the new task without altering the base model. Ladder side-tuning [29] hierarchically adds multiple side networks, with each side network responsible for modulating the activations of a specific layer in the pre-trained model. While READ takes inspiration from Side-Tuning and LST, we would like to highlight significant differences between READ and prior works. First, READ only contains a single RNN block which takes in the hidden state of the backbone network in a recurrent manner. This way, the number of parameters to fine-tune does not increase with the size of the backbone, whereas LST attaches multiple transformer blocks to the backbone network. When the backbone gets larger, the size of the LST network also gets larger. Secondly, Side-Tuning uses an additive side network to sum its representation with the backbone network in only the last layer. READ consumes the backbone's hidden state at every layer to iteratively calculate its RNN states. The recurrence nature of RNN allows information to flow from one layer to the next, which is why READ outperforms other PETL methods. Last, our fine-tuning is transformer-free as only RNN and Feed-Forward Network (FNN) structures are used in READ and require no transformer

Table 7: Model architectures for four different sized T5 models.

	Params (Million)	Encoder Layers	Decoder Layers	Heads	Embedding Dimension	Head Dimension	FFN Dimension
$T5_{SMALL}$	77	6	6	8	512	64	2048
$T5_{BASE}$	248	12	12	12	768	64	3072
$T5_{LARGE}$	771	24	24	16	1024	64	4069
$T5_{3B}$	2885	24	24	32	1024	128	16384

or attention mechanism. We may use a randomly initialized READ network without going through pre-training like in LST or exploiting any subtle tricks for training a transformer.

D Experimental Details

Energy Consumption Measurement. Higher training efficiency translates to lower energy consumption. To demonstrate the training efficiency benefit of READ, we measure and report the training GPU energy consumption (in kWh) for every experiment. We adopt the following commonly-used methodology to measure and estimate the model training energy consumption. We take the GPU resource utilization into account when computing the corresponding energy consumption by assuming a simple linear relationship between GPU utilization and its power consumption. Assume a training experiment endures H hours on GPUs, with power consumption of p_0 kW, at the GPU utilization level (summed over all GPU nodes) $u(t)$ (in percent). Then the total energy consumption (in kWh) is given by

$$E = \int_0^H \frac{u(t)}{100} \cdot p_0 dt = H \cdot \left(\frac{1}{H} \int_0^H u(t) dt \right) \cdot \frac{p_0}{100}. \tag{15}$$

In practice, we sample $u(t)$ at the granularity of minutes throughout training using NVIDIA's System Management Interface (smi). We then calculate its cumulative sum $\overline{U} = \sum_{i=1}^{60H} u_i$, thereby we can approximate the right hand side of Equation (15) by

$$H \cdot \frac{\sum_{i=1}^{60H} u_i}{60H} \cdot \frac{p_0}{100} = \overline{U} \cdot \frac{p_0}{6000}. \tag{16}$$

When reporting the energy consumption analysis for READ and other designs (see Section 4), we use $p_0 = 0.25$ kW for a NVIDIA V100 32 GB GPU [7] for Equation (16).

D.1 Dataset and model details

GLUE Datasets In Table 6, we list the dataset size, number of GPU nodes, and training batch size per GPU node for every task in GLUE. Note the total batch size (summed over all nodes) are fixed as 96 across all tasks and all methods.

T5 models Table 7 gives architecture-related numbers for four sizes of T_5 model. Note for all experiments $T5_{BASE}$ we use the original archtectures, while for READ experiments with $T5_{LARGE}$, we drop the last 4 layers from both encoder and decoder.

D.2 Hyperparameters

Architecture search For fine-tuning methods that have tunable architectural hyperparameters (e.g. RNN hidden dimensions in READ, ranks in LoRA, etc), we do hyperparameter search as follows: first, we fix the architecture $\mathcal{A}$ (e.g. in READ, take RNN-dim $= 128$ and side-net type to be LSTM), and do learning rate search for every dataset $\mathcal{D}$. Among each hyperparameter sweep $\mathscr{H}(\mathcal{D})$ there exists a run $\mathcal{R}^*(\mathcal{D})$ that has the best validation score $\mathscr{S}(\mathcal{D})$. Then we calculate the average of $\mathscr{S}(\mathcal{D})$ across all datasets $\mathcal{D}$ as the quality score of $\mathcal{A}$, denoted as $\mathscr{S}(\mathcal{A})$. Now we move on to the next architecture (e.g. in READ, take RNN-dim $= 256$ and side-net to be GRU) and repeat the above process. After iterating through all architecture candidates, we choose the architecture $\mathcal{A}^*$ that has

[7] 250W comes from the datasheet on NVIDIA's website

Table 8: Final archtecture choices for all PEFT experiments reported in Section 4.

	READ	READ large	Adapter	LoRA	Prompt tuning	LST
Architecture HP Names	RNN-type/ RNN-dim	RNN type/ RNN-dim	Bottleneck size	Rank	Number of prompts	Sidenet-dim
Architecture Candidates	{GRU/256, GRU/128, LSTM/128}	{GRU/256, GRU/128, LSTM/128}	{32, 64, 128}	{8, 16, 32}	{10, 20, 30}	{64, 96, 128}
Final Choices	GRU/256	GRU/256	64	32	20	96

Table 9: Final learning rates for all fine-tuning methods and GLUE datasets

	CoLA	MNLI	QNLI	MRPC	QQP	SST-2	STS-B
Full-tuning	9×10^{-6}	7.16×10^{-5}	3.76×10^{-4}	3.59×10^{-5}	1.75×10^{-4}	4.6×10^{-6}	1.30×10^{-4}
Adapter	1.16×10^{-3}	7.47×10^{-4}	4.6×10^{-6}	1.95×10^{-3}	4.6×10^{-6}	1.46×10^{-4}	2.83×10^{-3}
LoRA	1.75×10^{-4}	3.05×10^{-5}	9×10^{-6}	1.75×10^{-4}	9×10^{-6}	7.16×10^{-5}	1.15×10^{-4}
BitFit	3×10^{-3}	2.83×10^{-3}	2.83×10^{-3}	2.83×10^{-3}	2.83×10^{-3}	3×10^{-3}	2.83×10^{-3}
Prompt-tuning	2.83×10^{-3}	1.40×10^{-3}	7.47×10^{-4}	3×10^{-3}	2.83×10^{-3}	3×10^{-3}	2.74×10^{-3}
LST	2.51×10^{-4}	1.75×10^{-4}	7.16×10^{-5}	7.47×10^{-4}	3.7×10^{-4}	1.75×10^{-4}	1.45×10^{-3}
READ	3.29×10^{-4}	3.67×10^{-4}	1.75×10^{-4}	7.8×10^{-5}	1×10^{-6}	2.5×10^{-6}	4.6×10^{-5}
READ-large	8.5×10^{-5}	1.46×10^{-4}	1.75×10^{-4}	1.43×10^{-3}	2.13×10^{-4}	2.04×10^{-4}	7.1×10^{-5}

the best score $\mathscr{S}(\mathcal{A}^*)$, and report the test scores of each best run $\mathcal{R}^*(\mathcal{D})$ of $\mathcal{A}^*$. Therefore, each method in Table 1 adopts the same architectures throughout all datasets. For Full-tuning and BitFit where no architectural hyperparameters are present, we do the learning rate search once to obtain the test scores.

Learning rate search For each learning rate sweep, we do learning rates search in between $[1 \times 10^{-6}, 3 \times 10^{-3}]$ at log-scale for up to 32 rounds, where we employ Bayesian optimization for faster convergence of hyperparameter sweeps at lower computation costs.

Hyperparameter choices Table 8 and 9 summarize our final choices of architectural hyperparameters and learning rates.