Dynamic Subset Tuning: Expanding the Operational Range of Parameter-Efficient Training for Large Language Models

Felix Stahlberg, Jared Lichtarge, Shankar Kumar
Google Research
{fstahlberg,lichtarge,shankarkumar}@google.com

Abstract

We propose a novel parameter-efficient training (PET) method for large language models that adapts models to downstream tasks by optimizing a small subset of the existing model parameters. Unlike prior methods, this subset is not fixed in location but rather which parameters are modified evolves over the course of training. This dynamic parameter selection can yield good performance with many fewer parameters than extant methods. Our method enables a seamless scaling of the subset size across an arbitrary proportion of the total model size, while popular PET approaches like prompt tuning and LoRA cover only a small part of this spectrum. We match or outperform prompt tuning and LoRA in most cases on a variety of NLP tasks (MT, QA, GSM8K, SuperGLUE) for a given parameter budget across different model families and sizes.

1 Introduction

Large language models (LLMs) such as GPT-4 [28] and PaLM [1, 6] have demonstrated remarkable performance on various natural language processing (NLP) tasks. A common paradigm for adapting LLMs to specific NLP tasks is fine-tuning the pre-trained model on a downstream task dataset. Effective regularization is paramount since fine-tuning is often prone to overfitting due to the large model size and the small number of examples in the fine-tuning dataset. Furthermore, storing a full model version for each task becomes prohibitive as the number of tasks grows. Parameter-efficient training (PET) [9] is a family of approaches that regularizes fine-tuning by allowing only a small number of parameters to change, thus reducing the storage overhead for each task drastically. Popular techniques such as prompt tuning [24] and LoRA [18] fall into this category.

A recent line of work in PET keeps the internal structure of the seed model intact by identifying a small subset of parameters that is optimized in fine-tuning while freezing all others. Ding et al. [9] refer to this as *specification-based delta tuning*. Typically, these approaches specify the free parameter set *once* either before or after model adaptation. The subset is chosen based on either heuristics [3], Fisher information [47, 42], or the fine-tuning training signal [27, 13, 2, 30].

In this work, we propose a novel algorithm called *dynamic subset tuning* (DST) that efficiently re-selects the set of free parameters *in each training step* without a substantial slowdown in training. Our method is more flexible than prompt tuning and LoRA as it allows us to precisely specify the fine-tuning parameter budget as a fraction of the model size. Our method often matches or outperforms prompt tuning and LoRA with comparable parameter budget on various NLP tasks, and covers a much wider range of subset sizes. In particular, it enables the use of PET with many fewer free parameters (down to 0.00001% of the network) than most prior work, allowing DST to make use of even very small training sets.

2 Dynamic subset tuning

Let $\Theta^{(t)} \in \mathbb{R}^n$ be the parameter vector at time step t, where n is the number of parameters in the model. $\Theta^{(0)} \in \mathbb{R}^n$ refers to the parameters of the pre-trained seed model. In DST, the fine-tuned parameter vector differs from the seed model in exactly ϵn components ($\epsilon \in (0,1)$). This constraint can be enforced at different granularity levels, which we describe using a partition $\mathcal{S}$ of $[1,n]$ into k "silos". $\mathcal{S} = \{S_i\}_{i=1}^k$ where each $S_i \subseteq [1,n]$, and $\bigcup_{i=1}^k S_i = [1,n]$, for any $i \neq j$, $S_i \bigcap S_j = \emptyset$, and $\sum_{i=1}^k |S_i| = n$ (i.e. a mutually exclusive and jointly exhaustive division of the model parameters). In this paper, we use **per-module-and-layer siloing**: Each module in each layer is a separate silo that has a constant fraction of free parameters, but the set of free parameters can still be spread out within the silo (e.g. within a weight matrix). Appendix A compares this siloing strategy with alternative approaches. We denote the set of valid parameter vectors as $\mathcal{H}_{\epsilon,\mathcal{S}}$:

$$\mathcal{H}_{\epsilon,\mathcal{S}} := \{\Theta \in \mathbb{R}^n | \forall S \in \mathcal{S} : \epsilon|S| = \sum_{i\in S} I(\Theta_i^{(0)} \neq \Theta_i)\}. \tag{1}$$

where Θ_i is the i^{th} component of the parameter vector Θ and $I(\cdot)$ is the indicator function (1 if the condition is true, 0 otherwise). The ϵ-constraint is enforced in each silo separately:

$$\forall t \in \mathbb{N} : \Theta^{(t)} \in \mathcal{H}_{\epsilon,\mathcal{S}}. \tag{2}$$

The silo-level ϵ-constraints in Eq. 2 imply that, at each time step t, the number of parameters across the *full* model that differ from $\Theta^{(0)}$ is exactly ϵn since $\mathcal{H}_{\epsilon,\mathcal{S}} \subset \mathcal{H}_{\epsilon,\{[1,n]\}}$. We denote the updates computed using the optimizer's update rule with $u^{(t)} \in \mathbb{R}^n$ such that a full update step to $\hat{\Theta}^{(t+1)} \in \mathbb{R}^n$ can be described as:

$$\hat{\Theta}^{(t+1)} = \Theta^{(t)} + u^{(t)}. \tag{3}$$

Vanilla fine-tuning would use $\hat{\Theta}^{(t+1)}$ directly in the next training iteration ($\Theta^{(t+1)} \overset{\text{Full-FT}}{=} \hat{\Theta}^{(t+1)}$). In DST, however, we aim to find a $\Theta^{(t+1)}$ that is *close* to the fully updated $\hat{\Theta}^{(t+1)}$ but satisfies Eq. 2:

$$\Theta^{(t+1)} = \underset{\Theta\in\mathcal{H}_{\epsilon,\mathcal{S}}}{\arg\min} \sum_{i=1}^n d(\Theta_i, \hat{\Theta}_i^{(t+1)}, \Theta_i^{(0)}), \tag{4}$$

where $d(\cdot,\cdot,\cdot)$ is a distance function. An example is the component-wise **inverse-relative distance function** that biases towards both large differences and seed parameter magnitudes:

$$d(\Theta_i, \hat{\Theta}_i^{(t+1)}, \Theta_i^{(0)}) = |\Theta_i^{(0)}(\Theta_i - \hat{\Theta}_i^{(t+1)})| \tag{5}$$

Appendix A compares different variants of this distance function. Eq. 4 can be solved for each silo $S \in \mathcal{S}$ separately by using values from $\hat{\Theta}^{(t+1)}$ in the top-$\epsilon|S|$ positions of the distance vector between $\Theta^{(0)}$ and $\hat{\Theta}^{(t+1)}$, and resetting the remaining values to $\Theta^{(0)}$. More formally, let Δ be the distance vector and $q_S \in \mathbb{R}_+$ be the threshold that separates the top-$\epsilon|S|$ distances from the rest:

$$\forall i \in [1,n] : \Delta_i = d(\Theta_i^{(0)}, \hat{\Theta}_i^{(t+1)}, \Theta_i^{(0)}) \text{ and } \epsilon|S| = \sum_{i\in S} I(\Delta_i > q_S). \tag{6}$$

We construct the final parameter vector $\Theta^{(t+1)}\big|_S$ for silo S as follows:

$$\forall i \in S : \Theta_i^{(t+1)} = \begin{cases} \hat{\Theta}_i^{(t+1)}, & \text{if } \Delta_i > q_S \\ \Theta_i^{(0)}, & \text{otherwise} \end{cases}. \tag{7}$$

Alg. 1 shows a full update step in DST. [1]

3 Experimental setup

All our training runs start off from publicly available pre-trained checkpoints. For the foundation model PaLM 2 [1] we use the three sizes available via the Google Cloud API: **Gecko**, **Otter**, and **Bison**. For T5 we use the T5 1.1 checkpoints[2] available in T5X [36]: **Small** (77M parameters), **Base**

[1]In practice, we use an iterative approximation to the `quantile()` function in line 5 which makes the computational overhead negligible. See Appendix C for more details.

[2]`https://github.com/google-research/text-to-text-transfer-transformer/blob/main/released_checkpoints.md`

Algorithm 1 DST update function for computing the model parameters $\Theta^{(t+1)}$ for the next training iteration. Note that `compute_full_updates()` may have additional dependencies such as the optimizer state in momentum-based optimizers that we left out for the sake of simplicity.

```
Require: $\Theta^{(0)}$: Seed params; $\Theta^{(t)}$: Params at time step $t$; $\epsilon$: Fraction of free params; $\mathcal{S}$: Siloing partition
1: $u \leftarrow \texttt{compute\_full\_updates}(\Theta^{(t)})$ {Apply optimizer's update rule}
2: $\hat{\Theta} \leftarrow \Theta^{(t)} + u$
3: $\Delta \leftarrow d(\Theta^{(0)}, \hat{\Theta}, \Theta^{(0)})$ {Component-wise}
4: for $S \in \mathcal{S}$ do
5:     $q \leftarrow \texttt{quantile}(\Delta\big|_S, 1 - \epsilon)$
6:     $\Theta^{(t+1)}\big|_S \leftarrow \texttt{where}(\Delta\big|_S > q, \hat{\Theta}\big|_S, \Theta^{(0)}\big|_S)$
7: end for
8: return $\Theta^{(t+1)}$
```

(250M parameters), **Large** (800M parameters), **XL** (3B parameters), and **XXL** (11B parameters). AdaFactor [40] state variables like momentum are computed using the fully updated parameters $\hat{\Theta}^{(t)}$, not the post-processed ones $\Theta^{(t)}$. The dropout rates and learning rates are tuned on the respective validation sets – see Appendix F for details. We test our method on the following benchmarks: Machine Translation (WMT22 German-English), Trivia-QA (closed-book, open-domain question answering), GSM8K (math problems), SuperGLUE (eight language understanding tasks). Appendix G lists more details about the datasets and our evaluation protocols.

4 Results

4.1 Comparison with prompt tuning and LoRA

We compare DST with two popular PET baselines: **Prompt tuning** [24] is an extension of prompt engineering. It replaces hard-coded textual prompts with soft prompt-embeddings that are trained on the downstream task. **LoRA** [18] adds linear bottleneck structures to the network that are parallel to the attention weight matrices. All parameters besides the ones in the added bottlenecks are frozen during fine-tuning.

Fig. 1 compares both PET methods with DST as a function of the number of free parameters. We show the performance of the PET methods relative to the respective fully fine-tuned model.[3] We vary the prompt length in prompt tuning between 1 and 100. LoRA ranks range between 1 and 50. Like Hu et al. [18] we noticed that higher LoRA ranks often do not yield further gains, possibly due to regularization being less effective. Similarly, prompt tuning peaks at a prompt length of 10 ($\approx 10^{-5}$ free parameter fraction). Compared to LoRA and prompt-tuning, DST curves (red) show good performance across a broader range of the allocated parameter budget. This shows that there is more flexibility in controlling the parameter budget. On very small training sets (Fig. 1e), DST with the Otter model results in a smooth performance curve, while other systems are either erratic (prompt tuning) or stay close to the chance level.

4.2 T5 experiments

Fig. 2 shows the trade-off between the quality and the subset size for different T5 model sizes. For $\epsilon \geq 10^{-4}$ (i.e. 0.01% free parameters) the performance of all our T5 models except *Small* stays within 3% of the full fine-tuning baseline. Smaller models are more affected by an even lower ϵ than larger models. This suggests that the system performance mainly depends on the absolute size of the subset: *Small* is more than 100x smaller than *XXL*, and so are the absolute sizes of their subsets for the same ϵ.

4.3 Multi-subset serving latency

In addition to regularization, a common use case for PET is model sharing. For example, in DST, a single base model can be adapted to the task at hand by swapping out the subset at inference time. Fig. 3 shows that the latency increase of applying DST parameter subsets on-the-fly is close or below

[3]Appendix D lists the absolute scores of all systems.

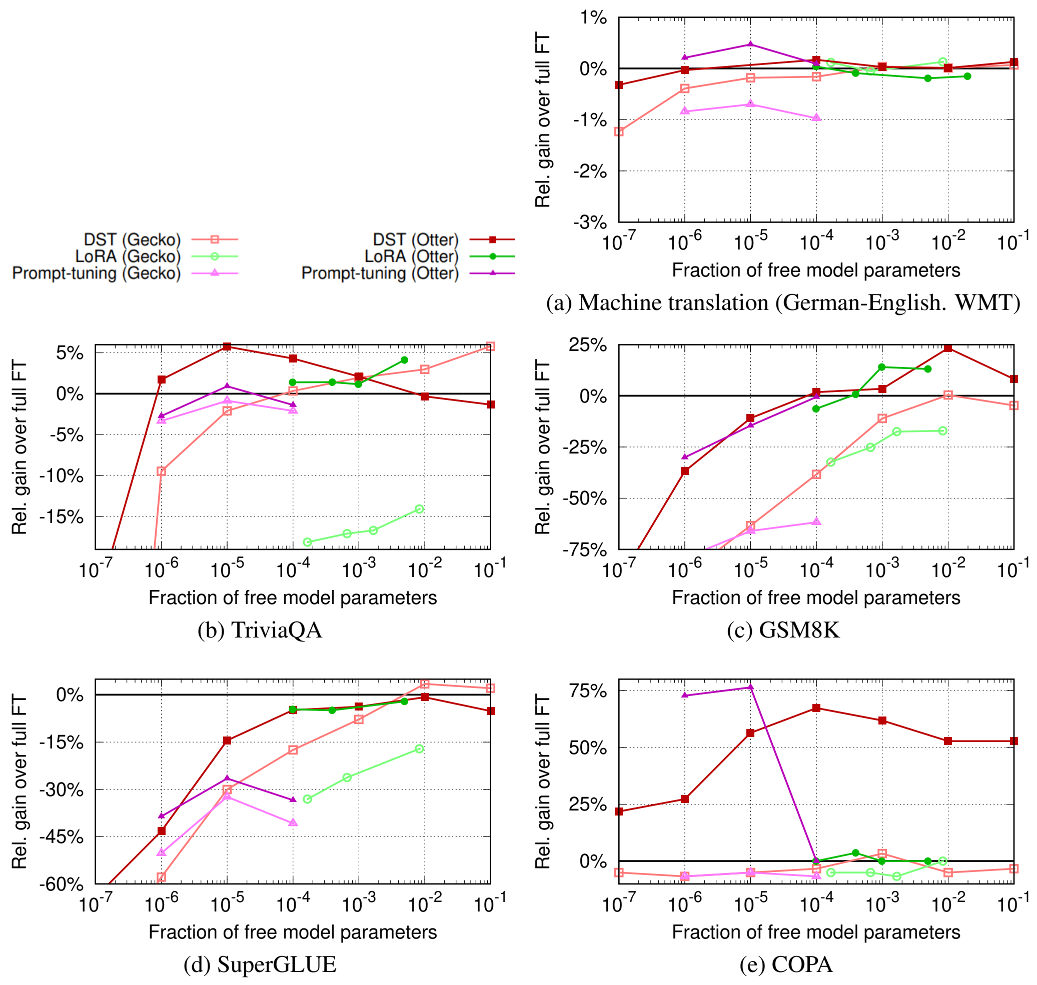


Figure 1: Validation set performance of the Gecko and Otter models as a function of the fraction of free parameters relative to full fine-tuning.

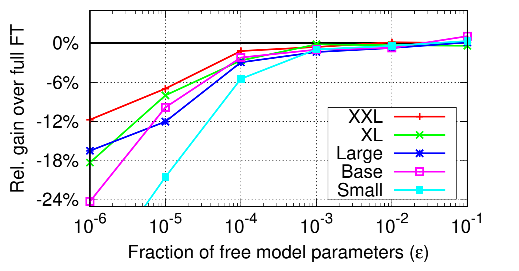


Figure 2: Average scores of T5 models on SuperGLUE compared to full fine-tuning as a function of ϵ.

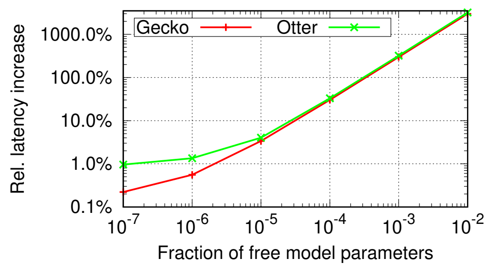


Figure 3: Latency increase of applying subset weights on-the-fly before decoding as a function of ϵ. We use the MT test sentence “Wie geht es dir heute?”.

1% for $\epsilon \leq 10^{-6}$ for a basic JAX-based implementation on a 2x2 TPU v5e, potentially making network-based client-server scenarios viable in a scenario where the subset is sent by the client. Our multi-subset inference implementation is not suitable for efficiently serving subsets larger than $\epsilon = 10^{-4}$, but better support for sparse matrix operations[4] may improve the efficiency in the future.

[4]`https://jax.readthedocs.io/en/latest/jax.experimental.sparse.html`

Note that there is no latency increase for single-subset serving as the subset can be embedded in the model at startup time.

5 Related work

Besides LoRA and prompt tuning, DST is closely related to delta tuning [9] approaches such as *diff pruning* [27, 13]. Diff pruning uses a continuous relaxation of the subset constraint which is determinized *once* after training. Other delta tuning methods choose the subnetwork mask *prior* to fine-tuning based on the Fisher information [47, 42]. In contrast, DST re-selects the subset in every step. DST is more flexible than heuristic-based delta tuning approaches like BitFit [3] since the subset is learnt. Our absolute distance function in Table 1 is inspired by Ansell et al. [2]. We extend prior work on delta tuning by (a) proposing an algorithm that allows for discrete subset churn at each training step, (b) introducing the concept of siloing, and (c) comparing distance normalizing strategies. We show on a number of benchmarks that DST covers a wider range of subset sizes than our baselines.

DST uses the insight that a small subnetwork is sufficient for adapting a model to a downstream task. This is connected to the lottery ticket hypothesis [11] that states that subnetworks can reach test accuracy comparable to the original network. Similarly, Zhang et al. [50] suggested that the activations in vanilla Transformers tend to be sparse.

Delta tuning approaches such as DST are related to model pruning. Similar to movement pruning [38], our distance functions in Table 1 rely on changes in weights. Our inverse-relative distance function is motivated by magnitude pruning [15]. Although intuitively similar, DST's goal is PET and not model shrinking.

Adapters [34, 17, 31, 16] are small bottleneck layers inserted into the pre-trained model that are learnt in fine-tuning while the rest of the model is frozen. Unlike adapters, DST leaves the network structure untouched.

Su et al. [41] and Xu and Zhang [46] use random masks to reduce the number of free parameters in fine-tuning. We compare DST with random masking in Sec. A. Koohpayegani et al. [22] reduce the number of parameters in LoRA by factoring the low rank matrices using random basis matrices. By contrast, DST does not require a low rank factorization.

We demonstrated that DST has a regularization effect that yields gains over full fine-tuning when training data is scarse. This confirms prior work [44, 26, 29] showing that PET can outperform full fine-tuning. Regularizing training by limiting updates to a subset of the model parameters has also been explored by Lee et al. [23] and Zhang et al. [49]. Similar ideas inspired techniques like block coordinate descent or stacking [12] that aim to improve the efficiency and scalability of training large models. In these methods, unlike DST, parameters are not reset to their seed value when they drop out of the subset, resulting in a final model that differs more from the seed and has a larger effective subset size. Elastic weight consolidation [20, EWC] and DST both aim to regularize by keeping parameters close to their seed values, but EWC allows *all* parameters to change. DST is less expensive than EWC since it does not require computing statistics such as the Fisher information over the pre-training dataset.

6 Conclusion

In this work, we proposed a novel method for PET, Dynamic Subset Tuning (DST). DST is more flexible than extant PET methods, as it *simultaneously* optimizes both the subset of parameters to update and their corresponding values, and dynamically re-selects that subset every training step. We showed that because the size of the updated subset can be arbitrarily set, smaller subsets can be trained than with other methods, yielding a smaller PET model size. We showed that DST matches or outperforms LoRA and prompt tuning on a suite of NLP tasks and across various model types and sizes.

Our appendix includes analysis that sheds light on the function and application of DST, investigating the impact of churn, siloing, subset generalization, and computational costs. The limitations of our work are listed in Appendix H.

References

[1] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

[2] A. Ansell, E. Ponti, A. Korhonen, and I. Vulić. Composable sparse fine-tuning for cross-lingual transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1778–1796, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[3] E. Ben-Zaken, Y. Goldberg, and S. Ravfogel. 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, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[4] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[5] G. Chen, F. Liu, Z. Meng, and S. Liang. Revisiting parameter-efficient tuning: Are we really there yet? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2612–2626, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics.

[6] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[7] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[8] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer. 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*, 2021.

[9] N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, Y. Chen, C.-M. Chan, W. Chen, et al. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*, 2022.

[10] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 06–11 Aug 2017.

[11] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

[12] L. Gong, D. He, Z. Li, T. Qin, L. Wang, and T. Liu. Efficient training of BERT by progressively stacking. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2337–2346. PMLR, 09–15 Jun 2019.

[13] D. Guo, A. Rush, and Y. Kim. Parameter-efficient transfer learning with diff pruning. 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 4884–4896, Online, Aug. 2021. Association for Computational Linguistics.

[14] B. Han, Y. Wu, G. Hu, and Q. Chen. Lan-bridge MT's participation in the WMT 2022 general translation shared task. In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 268–274, Abu Dhabi, United Arab Emirates (Hybrid), Dec. 2022. Association for Computational Linguistics.

[15] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[16] R. He, L. Liu, H. Ye, Q. Tan, B. Ding, L. Cheng, J. Low, L. Bing, and L. Si. On the effectiveness of adapter-based tuning for pretrained language model adaptation. In C. Zong, F. Xia, W. Li, and R. Navigli, editors, *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 2208–2222, Online, Aug. 2021. Association for Computational Linguistics.

[17] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for NLP. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR, 09–15 Jun 2019.

[18] E. J. Hu, yelong shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[19] M. Joshi, E. Choi, D. Weld, and L. Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[20] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[21] T. Kocmi, R. Bawden, O. Bojar, A. Dvorkovich, C. Federmann, M. Fishel, T. Gowda, Y. Graham, R. Grundkiewicz, B. Haddow, R. Knowles, P. Koehn, C. Monz, M. Morishita, M. Nagata, T. Nakazawa, M. Novák, M. Popel, and M. Popović. Findings of the 2022 conference on machine translation (WMT22). In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 1–45, Abu Dhabi, United Arab Emirates (Hybrid), Dec. 2022. Association for Computational Linguistics.

[22] S. A. Koohpayegani, K. Navaneet, P. Nooralinejad, S. Kolouri, and H. Pirsiavash. Nola: Compressing lora using linear combination of random basis. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[23] C. Lee, K. Cho, and W. Kang. Mixout: Effective regularization to finetune large-scale pretrained language models. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

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

[25] B. Li, J. Chen, and J. Zhu. Memory efficient optimizers with 4-bit states. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 15136–15171. Curran Associates, Inc., 2023.

[26] H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. A. Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 1950–1965. Curran Associates, Inc., 2022.

[27] A. Mallya, D. Davis, and S. Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European conference on computer vision (ECCV)*, pages 67–82, 2018.

[28] OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[29] S. Oymak, A. S. Rawat, M. Soltanolkotabi, and C. Thrampoulidis. On the role of attention in prompt-tuning. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 26724–26768. PMLR, 23–29 Jul 2023.

[30] A. Panigrahi, N. Saunshi, H. Zhao, and S. Arora. Task-specific skill localization in fine-tuned language models. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 27011–27033. PMLR, 23–29 Jul 2023.

[31] J. Pfeiffer, A. Kamath, A. Rücklé, K. Cho, and I. Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In P. Merlo, J. Tiedemann, and R. Tsarfaty, editors, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online, Apr. 2021. Association for Computational Linguistics.

[32] M. Post. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels, Oct. 2018. Association for Computational Linguistics.

[33] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1), jan 2020.

[34] S.-A. Rebuffi, H. Bilen, and A. Vedaldi. Learning multiple visual domains with residual adapters. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[35] A. Roberts, H. W. Chung, A. Levskaya, G. Mishra, J. Bradbury, D. Andor, S. Narang, B. Lester, C. Gaffney, A. Mohiuddin, C. Hawthorne, A. Lewkowycz, A. Salcianu, M. van Zee, J. Austin, S. Goodman, L. B. Soares, H. Hu, S. Tsvyashchenko, A. Chowdhery, J. Bastings, J. Bulian, X. Garcia, J. Ni, A. Chen, K. Kenealy, J. H. Clark, S. Lee, D. Garrette, J. Lee-Thorp, C. Raffel, N. Shazeer, M. Ritter, M. Bosma, A. Passos, J. Maitin-Shepard, N. Fiedel, M. Omernick, B. Saeta, R. Sepassi, A. Spiridonov, J. Newlan, and A. Gesmundo. Scaling up models and data with `t5x` and `seqio`. *arXiv preprint arXiv:2203.17189*, 2022.

[36] A. Roberts, H. W. Chung, G. Mishra, A. Levskaya, J. Bradbury, D. Andor, S. Narang, B. Lester, C. Gaffney, A. Mohiuddin, C. Hawthorne, A. Lewkowycz, A. Salcianu, M. van Zee, J. Austin, S. Goodman, L. B. Soares, H. Hu, S. Tsvyashchenko, A. Chowdhery, J. Bastings, J. Bulian, X. Garcia, J. Ni, A. Chen, K. Kenealy, K. Han, M. Casbon, J. H. Clark, S. Lee, D. Garrette, J. Lee-Thorp, C. Raffel, N. Shazeer, M. Ritter, M. Bosma, A. Passos, J. Maitin-Shepard, N. Fiedel, M. Omernick, B. Saeta, R. Sepassi, A. Spiridonov, J. Newlan, and A. Gesmundo. Scaling up models and data with t5x and seqio. *Journal of Machine Learning Research*, 24(377):1–8, 2023.

[37] A. Roberts, C. Raffel, and N. Shazeer. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5418–5426, Online, Nov. 2020. Association for Computational Linguistics.

[38] V. Sanh, T. Wolf, and A. Rush. Movement pruning: Adaptive sparsity by fine-tuning. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20378–20389. Curran Associates, Inc., 2020.

[39] T. Sellam, D. Das, and A. Parikh. BLEURT: Learning robust metrics for text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7881–7892, Online, July 2020. Association for Computational Linguistics.

[40] N. Shazeer and M. Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018.

[41] Y. Su, C.-M. Chan, J. Cheng, Y. Qin, Y. Lin, S. Hu, Z. Yang, N. Ding, X. Sun, G. Xie, Z. Liu, and M. Sun. Exploring the impact of model scaling on parameter-efficient tuning. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15062–15078, Singapore, Dec. 2023. Association for Computational Linguistics.

[42] Y.-L. Sung, V. Nair, and C. A. Raffel. Training neural networks with fixed sparse masks. *Advances in Neural Information Processing Systems*, 34:24193–24205, 2021.

[43] D. Vilar, M. Freitag, C. Cherry, J. Luo, V. Ratnakar, and G. Foster. Prompting PaLM for translation: Assessing strategies and performance. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15406–15427, Toronto, Canada, July 2023. Association for Computational Linguistics.

[44] T. Vu, B. Lester, N. Constant, R. Al-Rfou', and D. Cer. SPoT: Better frozen model adaptation through soft prompt transfer. In S. Muresan, P. Nakov, and A. Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5039–5059, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[45] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

[46] J. Xu and J. Zhang. Random masking finds winning tickets for parameter efficient fine-tuning. *arXiv preprint arXiv:2405.02596*, 2024.

[47] R. Xu, F. Luo, Z. Zhang, C. Tan, B. Chang, S. Huang, and F. Huang. Raise a child in large language model: Towards effective and generalizable fine-tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9514–9528, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics.

[48] C. Zan, K. Peng, L. Ding, B. Qiu, B. Liu, S. He, Q. Lu, Z. Zhang, C. Liu, W. Liu, Y. Zhan, and D. Tao. Vega-MT: The JD explore academy machine translation system for WMT22. In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 411–422, Abu Dhabi, United Arab Emirates (Hybrid), Dec. 2022. Association for Computational Linguistics.

[49] H. Zhang, G. Li, J. Li, Z. Zhang, Y. Zhu, and Z. Jin. Fine-tuning pre-trained language models effectively by optimizing subnetworks adaptively. *Advances in Neural Information Processing Systems*, 35:21442–21454, 2022.

[50] Z. Zhang, Y. Lin, Z. Liu, P. Li, M. Sun, and J. Zhou. MoEfication: Transformer feed-forward layers are mixtures of experts. In S. Muresan, P. Nakov, and A. Villavicencio, editors, *Findings of the Association for Computational Linguistics: ACL 2022*, pages 877–890, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[51] J. Zhao, Z. Zhang, B. Chen, Z. Wang, A. Anandkumar, and Y. Tian. Galore: Memory-efficient LLM training by gradient low-rank projection. In *Forty-first International Conference on Machine Learning*, 2024.

Absolute	**Relative**	**Inverse-relative**
$d_{\text{abs}}(\theta_1, \theta_2, \theta^{(0)}) = \lvert\theta_1 - \theta_2\rvert$	$d_{\text{rel}}(\theta_1, \theta_2, \theta^{(0)}) = \lvert\frac{\theta_1 - \theta_2}{\theta^{(0)}}\rvert$	$d_{\text{inv-rel}}(\theta_1, \theta_2, \theta^{(0)}) = \lvert\theta^{(0)}(\theta_1 - \theta_2)\rvert$

Table 1: Parameter-wise distance functions used in this work to measure the difference between two models (θ_1 and θ_2) relative to the seed model ($\theta^{(0)}$).

	$\epsilon = 10^{-6}$	$\epsilon = 10^{-4}$
No siloing		
Random	71.45%	73.92%
Absolute – $d_{\text{abs}}(\cdot)$	71.20%	73.25%
Relative – $d_{\text{rel}}(\cdot)$	72.45%	74.04%
Inverse-relative – $d_{\text{inv-rel}}(\cdot)$	73.47%	74.06%
Per-module siloing		
Relative – $d_{\text{rel}}(\cdot)$	72.11%	74.08%
Inverse-relative – $d_{\text{inv-rel}}(\cdot)$	73.30%	73.78%
Per-module-and-layer siloing		
Relative – $d_{\text{rel}}(\cdot)$	72.68%	74.18%
Inverse-relative – $d_{\text{inv-rel}}(\cdot)$	73.83%	74.00%

Table 2: MT development set performance of the Gecko model in terms of BLEURT for different siloing and scoring strategies.

A Distance functions and siloing

We used the **inverse-relative distance function** and **per-module-and-layer siloing** in the main paper. We experimented with the following alternative siloing strategies:

- **No siloing**: The subset can be distributed freely across the full model.
- **Per-module siloing**: Each module in the network (attention, feed-forward, embeddings, etc.) receives the same fraction of free parameters, but the set can spread out across layers.

Furthermore, we explored different variants of the inverse-relative distance function (Table 1).

Table 2 compares DST performance using the various distance functions and siloing strategies. We confirm the findings of Su et al. [41] and Xu and Zhang [46] that even random parameter selection[5] works well for sufficiently large subsets ($\epsilon = 10^{-4}$). However, with $\epsilon = 10^{-6}$ (i.e. only 0.0001% of model parameters are free), the relative $d_{\text{rel}}(\cdot)$ distance and inverse-relative $d_{\text{inv-rel}}(\cdot)$ distance functions substantially outperform the random baseline. Per-module-and-layer siloing yields further gains. The BLEURT score differences between the methods are smaller for larger subsets ($\epsilon = 10^{-4}$).

Training stability Chen et al. [5] demonstrated that training with existing PET methods like LoRA and prompt tuning is often unstable. The top panel in Fig. 4 shows that the variance between training runs in DST depends on both distance function and siloing. Siloing and distance normalization (Relative and Inverse-relative) increase the average BLEURT and also reduce the variance compared to the absolute distance function (purple bar). The distribution of the free parameters across the network modules is most stable with the Relative distance function (green bar in the bottom panel in Fig. 4).

B Siloing alternatives

Siloing leads to the subset of free parameters being more evenly distributed across the full model, and thereby reduces the variability between training runs. We investigated two alternatives to siloing that aim to make $d(\cdot)$ more comparable across different modules:

- Size normalization: Divide $d(\cdot)$ by the number of parameters in the module/layer.
- Mean normalization: Divide $d(\cdot)$ by the mean of the absolute seed values in the module/layer.

Table 3 shows that siloing works more reliably than both strategies, especially with small ϵ.

[5]The random baseline randomly selects the free parameters once at the start of fine-tuning.

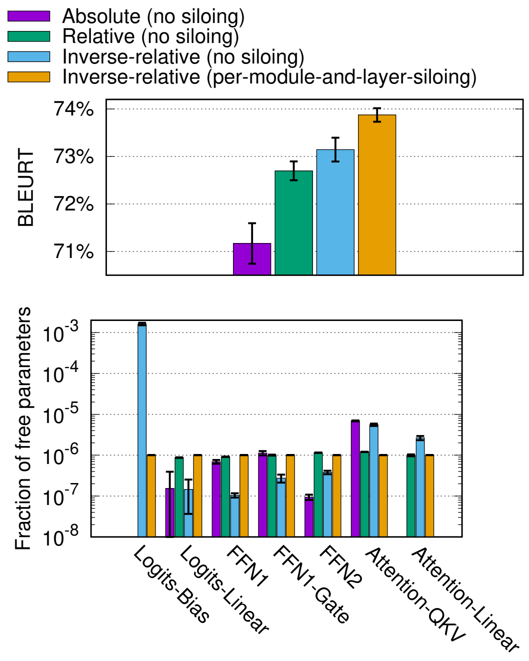


Figure 4: BLEURT scores and the fraction of free parameters per network module on the MT development set (Gecko model, $\epsilon = 10^{-6}$). Mean and standard deviation (error bars) across four training runs are shown.

Method	Per module		Per module and layer	
	$\epsilon = 10^{-6}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-6}$	$\epsilon = 10^{-4}$
Size normalization	72.66%	73.84%	73.19%	73.79%
Mean normalization	71.23%	73.25%	70.80%	73.76%
Siloing	73.30%	73.78%	73.83%	74.00%

Table 3: MT performance (BLEURT) of the Gecko model for alternatives to siloing.

C Efficient quantile computation

The DST update step (Alg. 1) involves computing the q_S threshold as the $(1-\epsilon)$-th quantile over the difference vector $\Delta\big|_S$. The standard implementation in JAX (`jax.numpy.quantile()`) is based on sorting the complete Δ vector which can be slow for large models. In fact, `jax.numpy.quantile()` does not support tensor sizes beyond 2^{32}, and is therefore not suitable for our larger models.

Fortunately, we noted that q_S does not change abruptly between training steps but rather fluctuates smoothly over time. Therefore, we use an iterative approach that refines the threshold from the previous training step (q') for a fixed number m of iterations. The algorithm maintains a lower and an upper bound that are initialized with $[\frac{q'}{r}, rq']$, i.e. we assume that q_S does not increase/decrease by more than a factor of r between training steps. Let $c(\cdot)$ be a function that counts the fraction of components in Δ greater than q:

$$c(\Delta, q) = \frac{1}{|\Delta|}\sum_{i=1}^{|\Delta|} I(\Delta_i > q). \tag{8}$$

The algorithm refines the lower and upper bounds for m steps by iteratively setting one of the bounds to the midpoint q_{mi} of the interval, depending on whether $c(\Delta, q_{mi})$ is less or greater than ϵ. Alg. 2 lists the full procedure. We found that the iterative method is able to track q_S very closely with $m = 3$ and $r = 2$. After an initial ramp up period of around 20 training steps, the relative error between the approximated $\tilde{\epsilon}$ (realized by $\tilde{q}$) and the exact value of ϵ usually stays below 1%.

Algorithm 2 Iterative refinement algorithm for approximating the threshold q_S for a single silo $S \in \mathcal{S}$.

```
Require: $q'$: The $q_S$ threshold from the previous training step.
Require: $\Delta$: Difference vector for $S$ of the current training step (see line 3 in Algorithm 1).
Require: $\epsilon$: Fraction of free parameters.
Require: $m$: Number of refinement iterations.
Require: $r$: Maximum fluctuation factor.
 1: $q_{lo} \leftarrow \frac{q'}{r}$
 2: $q_{mi} \leftarrow q'$
 3: $q_{hi} \leftarrow rq'$
 4: for $k \leftarrow 1$ to $m$ do
 5:     if $c(\Delta, q_{mi}) > \epsilon$ then
 6:         $q_{lo} \leftarrow q_{mi}$
 7:     else
 8:         $q_{hi} \leftarrow q_{mi}$
 9:     end if
10:     $q_{mi} \leftarrow \frac{q_{hi} - q_{lo}}{2}$
11: end for
12: return $\begin{cases} q_{lo}, & \text{if } |c(\Delta, q_{lo}) - \epsilon| < |c(\Delta, q_{hi}) - \epsilon| \\ q_{hi}, & \text{otherwise} \end{cases}$
```

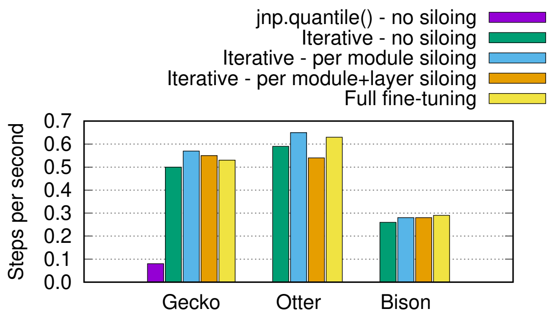


Figure 5: Training steps per second on TPU v4 chips with a batch size of 32. We use a 2x2x4 topology for the Gecko model and 4x4x4 for Otter and Bison.

Training speed Fig. 5 shows that a trivial implementation of DST using `jnp.quantile()` is much slower than full fine-tuning. However, using our novel approximate iterative algorithm to compute thresholds (Algorithm 2), we are able to match the full fine-tuning training speed. Algorithm 2 is crucial for making dynamic subset tuning with churn possible in practice even for large models (Bison).

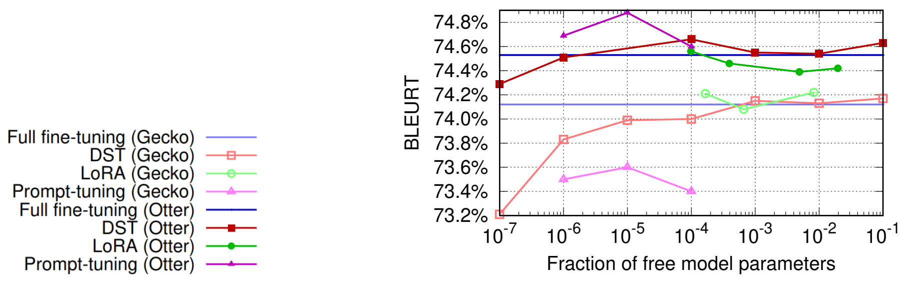
D Absolute performance metrics

Fig. 1 plots the performance curves of DST, LoRA, and prompt tuning relative to the respective fully fine-tuned baseline. Fig. 6 shows the same curves in absolute terms, marking the full fine-tuning baselines with blue (light: Gecko, dark: Otter) horizontal lines. The dark data series (Otter) outperform their lightly colored counter-parts (Gecko) because the Gecko model is 8x smaller than the Otter model.

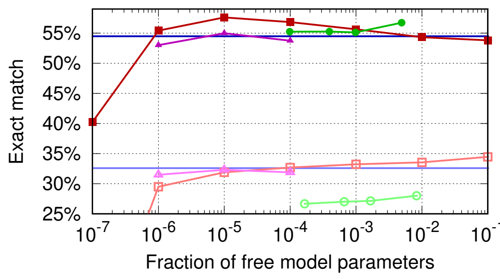
The average scores on SuperGLUE for the fully fine-tuned T5 baselines are listed in Table 4.

Table 5 shows that the BLEURT scores of our systems are comparable to the winners of the WMT22 evaluation campaign even though we used a very simple training pipeline (single fine-tuning stage on only 31.5K examples).

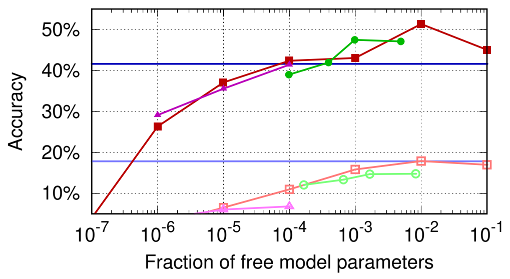
Table 6 shows the absolute BLEU and BLEURT scores on the WMT22 German-English test set for all PET techniques used in this work.



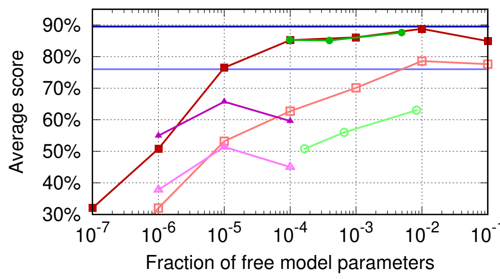
(a) Machine translation (German-English. WMT)



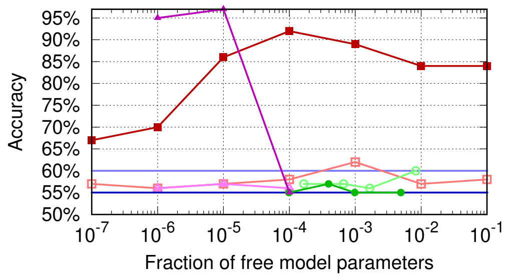
(b) TriviaQA



(c) GSM8K



(d) SuperGLUE



(e) CoPA

Figure 6: Validation set performance of the Gecko and Otter models as a function of the fraction of free parameters relative to full fine-tuning.

T5 model	Avg. score
T5-Small	86.3%
T5-Base	88.5%
T5-Large	89.9%
T5-XL	91.6%
T5-XXL	92.3%

Table 4: Average scores on SuperGLUE of the fully fine-tuned T5 baselines.

	BLEU	BLEURT
JDExploreA. [48]	49.3%	74.37%
Online-B	49.7%	74.00%
Lan-Bridge [14]	50.1%	73.84%
Online-A	50.2%	73.08%
DST (Bison model, $\epsilon = 10^{-6}$)	49.0%	74.13%

Table 5: Comparison with official German-English WMT22 evaluation systems [21].

E Subset overlap analyses

Churn A core difference between DST and other delta tuning methods is that it re-selects the parameter subset in each training step, i.e. it allows subset churn. In an ablation study we disabled churn by selecting the subset only once in the first training step. However, as shown in Table 7, churn helps the model to converge to a better subset, particularly when the subset is small ($\epsilon = 10^{-6}$).

To illustrate how the subset changes in the course of training we compute the *subset overlap* between two DST checkpoints as follows:

$$SubsetOverlap = \frac{|SubsetA \cap SubsetB|}{|SubsetA|}. \tag{9}$$

Base	0-shot	Full fine-tuning	Prompt-tuning	LoRA	This work (DST)		
			(length=10)	(rank=1)	$\epsilon = 10^{-7}$	$\epsilon = 10^{-6}$	$\epsilon = 10^{-4}$
				BLEU			
Gecko	47.0%	47.0%	43.8%	47.7%	46.8%	47.7%	47.2%
Otter	46.5%	47.3%	49.9%	46.9%	49.3%	49.5%	49.1%
Bison	47.9%	48.8%	48.4%	48.8%	49.5%	49.0%	48.4%
Avg.	**47.1%**	**47.7%**	**47.4%**	**47.8%**	**48.5%**	**48.7%**	**48.2%**
				BLEURT			
Gecko	71.39%	73.41%	73.01%	73.35%	72.23%	73.08%	73.18%
Otter	69.96%	73.56%	73.92%	73.53%	73.48%	73.73%	74.15%
Bison	72.70%	74.11%	74.18%	74.17%	73.90%	74.13%	74.21%
Avg.	**71.35%**	**73.69%**	**73.70%**	**73.68%**	**73.20%**	**73.65%**	**73.84%**

Table 6: BLEU and BLEURT scores of PET techniques on the WMT22 German-English test set.

ϵ	Siloing	No churn	With churn
10^{-6}	None	71.38%	73.47%
	Module	71.45%	73.30%
	Module+layer	72.55%	73.83%
10^{-4}	None	74.05%	74.06%
	Module	73.50%	73.78%
	Module+layer	73.69%	74.00%

Table 7: MT performance (BLEURT) of the Gecko model with and without subset churn.

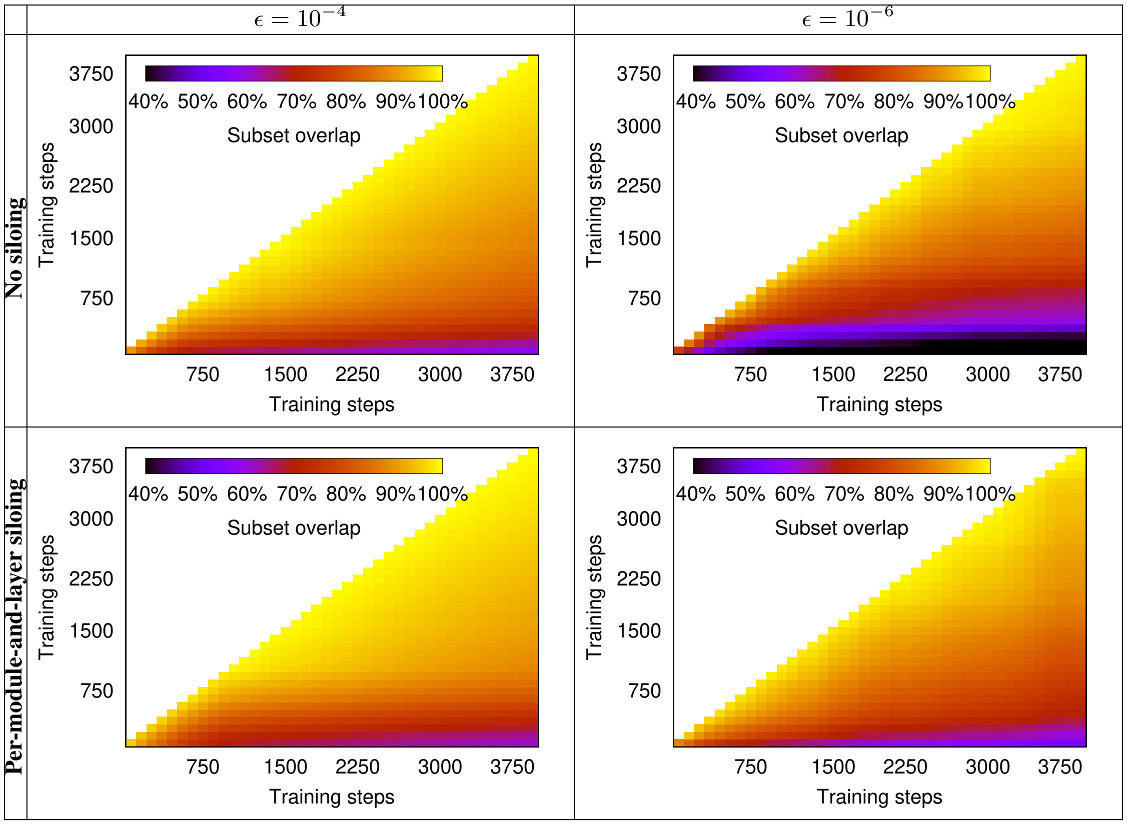
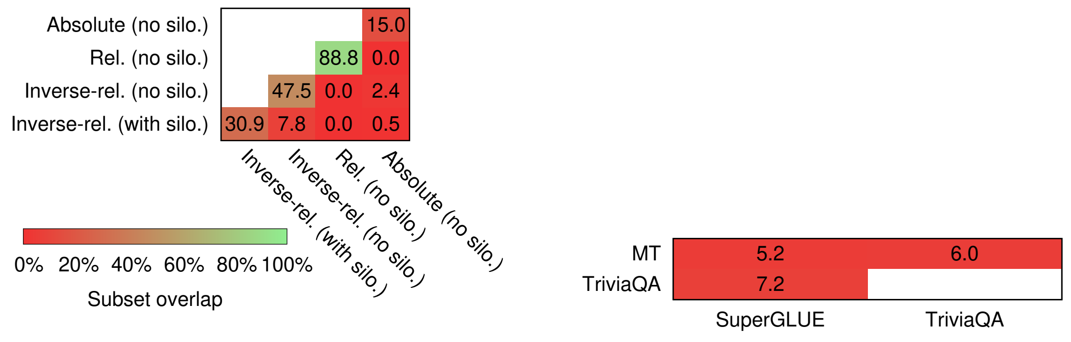


Figure 7: Subset overlaps between different training steps of the same training run for the Gecko model on MT.

Fig. 7 shows that the subset tends to converge slower without siloing and with a small ϵ. The overlap between the earliest and the latest checkpoint (bottom right corner in each diagram) is below 60% which indicates the importance of churn.



(a) Overlap (%) between different distance functions

(b) Overlap (%) between different tasks

Figure 8: Subset overlap averaged over four training runs (Gecko model, $\epsilon = 10^{-6}$, 3K training steps).

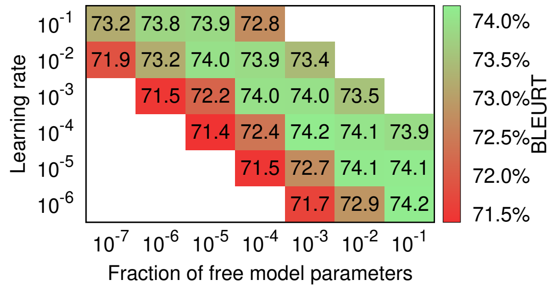


Figure 9: MT performance (BLEURT) as a function of the learning rate and the number of free parameters (ϵ).

Subset generalization A practical question is how well the learnt subset masks generalize to new tasks. The off-diagonal values in Fig. 8a indicate that there is very little subset overlap between different distance functions. The overlap stays below 50% even between training runs of the same configuration (diagonal in Fig. 8a) except for the Relative distance function. Subset overlaps between different tasks are relatively low, even when using the same distance function (Fig. 8b). This suggests that LLMs are highly redundant: they contain many different parameter subsets, each sufficient to adapt the model to a certain task. Combining DST with approaches like MAML [10] could potentially improve the generalization and make it possible to re-use subset masks across tasks.

F Training hyper-parameters

We train our models in the JAX [4] framework PAXML[6] on TPU v4 chips with AdaFactor [40]. We use a batch size of 32 for the PaLM 2 models and a batch size of 128 for T5. Dropout rates and learning rates are tuned for each experiment on the development sets. We used dropout for LoRA and full fine-tuning, but not for DST. We do not use dropout on GSM8K. For full fine-tuning, learning rates range between 0.1 and 1.0 in T5 and between 0.0001 and 0.000001 in PaLM 2.

Fig. 9 visualizes the inverse relationship between the optimal learning rate and ϵ in DST: the smaller the subset (i.e. the lower ϵ), the larger the optimal learning rate. Intuitively, a low ϵ resets a larger fraction of the parameters to the seed model, and thus reduces the total magnitude of the update step. A higher learning rate compensates for this effect. To reduce hallucinations in the PaLM 2 models we append the string "`\n[eod]`" to all training examples and truncate predictions after these tokens.

[6]`https://github.com/google/paxml`

Dataset		Size
WMT German-English		31.5K
Trivia-QA		87.6K
GSM8K		7.5K
SuperGLUE	BoolQ	9.4K
	CB	250
	COPA	400
	MultiRC	5.1K
	ReCoRD	101K
	RTE	2.5K
	WiC	6K
	WSC	554

Table 8: Number of examples in the training corpora.

G Datasets and evaluation protocols

We test our method on the following benchmarks: Machine Translation (WMT22 German-English), Trivia-QA (closed-book, open-domain question answering), GSM8K (math problems), SuperGLUE (eight language understanding tasks). Appendix G lists more details about the datasets and our evaluation protocols.

- **Machine translation (MT)**. LLMs such as PaLM have been successfully applied to MT [43]. Our main evaluation task is German-English translation using the official data from the WMT22 [21] evaluation campaign. We use `newstest2022` as the test set to avoid overlap with PaLM training data, a 500 sentence sample from `newstest2021` as the development set, and all previous WMT News test sets as the fine-tuning set. We report BLEURT[7] [39] scores against reference A and BLEU scores computed with SacreBLEU [32] against all available references.
- **Trivia-QA (closed-book, open-domain)** [37] is a version of the question answering task Trivia-QA [19] without direct access to the context of the question. We follow the seqio [35] setup[8] and report the number of exact matches.
- **GSM8K** [7] is a small dataset of high-quality grade school math word problems often used to assess LLMs.
- **SuperGLUE** [45] is a benchmark consisting of eight different language understanding tasks. Training set sizes range from 250 to 101K examples which we mix proportionally, following the `super_glue_v102_proportional`[9] recipe in T5 [33]. Like prior work we report average scores across all tasks, averaging scores of tasks with multiple scores first.

Table 8 lists the training data set sizes. We decode with beam search (beam size of 4) for MT, and with greedy search for the other benchmarks.

H Limitations

DST in its current form improves storage and regularization efficiency, but it does not improve the training efficiency or memory footprint. How to reduce the memory requirements in LLM adaptation is still an active area of research [8, 25, 51].

DST is a new approach for PET. Similar to existing PET methods such as prompt tuning and LoRA, our approach may be potentially used in a setting which can propagate or amplify existing biases from the dataset.

We tested DST with PaLM 2 and T5 models of different sizes, but we did not use other LLMs due to license restrictions and computational costs. Furthermore, we leave it to future work to explore the combination of multiple PET techniques.

[7] `BLEURT-20` checkpoint

[8] `https://github.com/google/seqio#triviaqa-closed-book-open-domain-version`

[9] `https://github.com/google-research/text-to-text-transfer-transformer/blob/main/t5/data/mixtures.py`