Sensitivity-Aware Adapter Placement for Memory-Aware Fine-Tuning of Transformer Encoders

Anonymous ACL submission

Abstract

Fine-tuning Transformer encoders under memory constraints is often limited by training-time GPU memory rather than compute alone. Standard LoRA instantiates adapters uniformly across a predefined module family, allocating trainable parameters and optimiser state even to modules that may contribute little to the downstream objective. We propose **Sensitivity-Aware Adapter Placement (SAAP)**, a method that decides *where* LoRA adapters should be instantiated before fine-tuning begins. SAAP scores candidate modules using a frozen-model sensitivity probe, then applies normalisation, adaptive cutoff selection, and stability filtering over repeated class-balanced subsamples to retain only a sparse, task-relevant adapter set.

We evaluate SAAP on encoder-based sequence classification using BERT-base, RoBERTa-base, and DistilBERT. Across all evaluated settings, SAAP uses substantially fewer trainable parameters than standard LoRA and often reduces both average and peak GPU memory while maintaining competitive predictive performance. The main empirical result is therefore a favourable memory–performance trade-off rather than uniform accuracy gains. We also compare against a budget-matched random sparse-placement baseline and show that SAAP often reduces end-to-end wall-clock runtime relative to standard LoRA and AdaLoRA, and remains trainable under severely constrained VRAM conditions where denser baselines fail.

1 Introduction

Large pretrained Transformers are frequently adapted under tight hardware, privacy, or latency constraints. In such settings, training-time GPU memory which is consumed by model weights, adapter tensors, gradients, optimiser state, activations required for backpropagation, and temporary buffers can be a binding constraint even for comparatively small models. Under uniform adapter placement, LoRA (Hu et al., 2022) allocates this overhead across all target modules, including those that may be only weakly relevant to the downstream task.

Parameter-efficient fine-tuning (PEFT) methods reduce adaptation cost by updating only a subset of parameters (Houlsby et al., 2019; Li and Liang, 2021; Hu et al., 2022). Among these, LoRA is widely used for its simplicity and lack of additional inference latency. However, standard LoRA addresses *how* updates are parameterised, not *where* limited adaptation capacity should be placed. Under strict memory budgets, this distinction matters: avoiding adapter instantiation in low-value modules also avoids their associated gradients and optimiser state. SAAP targets this structural placement problem and is therefore complementary to methods that focus on update parameterisation or rank allocation after the target-module set has already been fixed, such as AdaLoRA (Zhang et al., 2023) and quantised PEFT (Dettmers et al., 2023).

To decide which modules to adapt, SAAP uses a frozen-model sensitivity probe based on gradient- and activation-based importance estimation (Simonyan et al., 2014; Molchanov et al., 2019; Sanh et al., 2020), drawing on the observation that gradient-derived signals can identify where limited adaptation capacity is most task-relevant (Nowak et al., 2024). The scoring rule and its theoretical motivation are described in Section 2.

SAAP is not intended as a claim of uniform predictive superiority: its goal is a favourable memory–performance trade-off by reducing the number of instantiated adapter tensors and their associated gradients and optimiser state. On BERT-base/IMDb, for example, SAAP reduces trainable parameters from 1.19M to 0.026M and peak GPU memory from 3559 MB to 912 MB, at a cost of 2.5 accuracy points. The primary empirical focus is encoder-based sequence classification; Appendix H reports scope-broadening checks on TinyLlama-1.1B and

FLAN-T5-small as supplementary evidence only.

Our contributions are: (1) SAAP, a pre-fine-tuning sensitivity probe for sparse LoRA placement; (2) evidence across four encoder classification settings that SAAP reaches more favourable memory–performance operating points than LoRA and AdaLoRA with substantially fewer parameters; and (3) comparisons showing that SAAP usually outperforms budget-matched random placement, reduces end-to-end runtime, and remains trainable under severely constrained VRAM conditions where denser baselines fail.

2 Method

Let $\mathcal{M}$ denote the candidate module set, comprising the standard LoRA-style linear projections for each architecture family (query, value, and dense/output-style layers in attention and feed-forward blocks) and excluding embeddings and normalisation parameters. Task-specific classification head projections are included in $\mathcal{M}$ on the same basis, since their direct connection to the task loss makes them natural candidates for adaptation.

SAAP begins with a probing stage in which the backbone remains fully frozen; no parameter updates or optimiser state are maintained. The backward pass is used solely to compute sensitivity scores. For the sequence-classification tasks studied here, the task loss $\mathcal{L}$ is standard cross-entropy over target label classes. Let $A \in \mathbb{R}^{d_{\text{out}}}$ denote the output activation of a candidate module and $G = \partial\mathcal{L}/\partial A$ its loss gradient. We define the module score

$$S_{\text{GWM}}(m) = \sum_{x\in\mathcal{B}} \sum_{i} \left(|G_i^{(x)}|^2 \cdot |A_i^{(x)}| \right), \quad (1)$$

where $\mathcal{B}$ is a small class-balanced probe batch.

S_{GWM} can be interpreted as a local loss-sensitivity proxy. Gating module m's output by a scalar z_m and applying a first-order Taylor expansion gives

$$\Delta\mathcal{L}_m \approx \left| \left\langle \frac{\partial\mathcal{L}}{\partial A}, A \right\rangle \Delta z_m \right|,$$

linking loss change to an activation–gradient product, consistent with prior Taylor-based pruning criteria (Molchanov et al., 2017, 2019).

Squaring the gradient yields a Fisher-inspired variant: under diagonal approximations, $\mathbb{E}[G^2]$ approximates the diagonal Fisher, which has been used in prior work to estimate the cost of removing network components (Liu et al., 2021; Benzing, 2022). The combined form $|G|^2 \cdot |A|$ is therefore a practical hybrid. Taylor reasoning motivates $|G \cdot A|$, while Fisher reasoning motivates $\mathbb{E}[G^2]$. This biases selection towards modules both engaged by the input and influential for the task loss. We treat S_{GWM} as a computationally cheap proxy for this purpose, not as a theoretically optimal criterion.

Raw scores can be noisy and depth-dependent, so the remaining pipeline stages improve comparability and stability. First, scores are min–max normalised within each Transformer block to reduce depth-related scale drift. Second, rankings are computed independently for each target label class $c \in \mathcal{C}$, where "class" refers to downstream prediction labels rather than architectural module types; this prevents a single pooled ranking from being dominated by broadly useful modules at the expense of those strongly salient for only a subset of classes. Third, SAAP applies Kneedle (Satop"a"a et al., 2011) to each class-specific score curve to detect the point of maximum perpendicular deviation from the score–rank chord, yielding an adaptive cutoff k_c rather than a fixed global budget (see Appendix A for implementation details).

Fourth, the entire process is repeated over $N = 20$ stability passes using fold-style subsampling without replacement within each pass; only modules selected in every pass are retained. In our default configuration we use a 99% retention threshold, which with $N = 20$ corresponds to unanimous retention. Preliminary checks showed that looser thresholds produced variable module sets and unstable downstream behaviour, while larger N offered no benefit at higher probing cost. The final adapter set is the union across class-specific selections,

$$\mathcal{M}_{\text{final}} = \bigcup_{c\in\mathcal{C}} \text{Top}_{k_c}(\mathcal{M}),$$

which preserves coverage of class-specific salience patterns that a single pooled ranking would suppress, at the cost of reduced sparsity in many-class tasks.

3 Experiments and Results

We evaluate SAAP primarily on encoder-based sequence classification using BERT-base, RoBERTa-base, and DistilBERT on IMDb, AG News, Yelp Polarity, and TweetEval Hate. Appendix H reports limited scope-broadening checks on a decoder-only model (TinyLlama-1.1B) and an encoder–decoder

model (FLAN-T5-small). Baselines are full fine-tuning, standard LoRA, AdaLoRA, and a budget-matched random sparse-placement baseline. All experiments use 5,000 training and 1,000 evaluation examples, 4 epochs, effective batch size 32, learning rate $5\mathrm{e}{-4}$, and weight decay 0.01. For LoRA-based methods, we use rank $r = 8$, $\alpha = 16$, and dropout 0.05; AdaLoRA uses initial rank $r = 12$ and target rank $r = 8$.[1] SAAP uses the GWM score and a fixed, class-balanced probe pool that samples the same number of training examples from each label class. In the default configuration this corresponds to 50 probe examples per class (100 total for binary tasks and 200 total for 4-way AG News). We perform $N = 20$ stability passes, each pass using a class-balanced subsample drawn from this fixed pool without replacement within the pass, and retain only modules that are selected in at least 99% of passes (which, with $N = 20$, corresponds to being selected in every pass). Results are averaged over 5 seeds $\{45, 46, 47, 48, 49\}$. GPU memory is reported as average and peak allocated CUDA memory during downstream fine-tuning only; the SAAP probing stage is excluded, so SAAP's true worst-case memory may be modestly higher than the figures shown.

Table 1 shows a consistent memory–performance trade-off rather than uniform accuracy gains. SAAP lies on the score/peak-memory Pareto frontier in all four encoder settings: no other method simultaneously achieves higher score and lower peak GPU footprint. Relative to LoRA, SAAP reduces trainable parameters by a mean of 68.3% and peak GPU memory by 60.4%; relative to AdaLoRA, the reductions are 73.6% and 62.8%.

The comparison against budget-matched random placement is the clearest test of whether SAAP contributes beyond sparsity alone. Across the four main encoder settings, SAAP improves over random placement in score on 3/4 tasks and achieves lower peak GPU memory on all four. The only score reversal occurs on DistilBERT/Yelp Polarity, where Random scores 0.929 ± 0.005 against SAAP's 0.926 ± 0.002; as the gap of 0.003 is smaller than the standard deviation of either method, this should not be read as a reliable difference. Random placement also exhibits substantially higher variance in several settings: standard deviations of 0.037, 0.043, and 0.027 on BERT-base/AG News, BERT-base/IMDb, and RoBERTa-base/TweetEval Hate respectively, compared with 0.004, 0.001, and 0.001 for SAAP under the same parameter budgets. This pattern is consistent with uninformed sparse placement being high-variance: performance depends strongly on whether the sampled modules happen to overlap with task-relevant computation.

Appendix H reports limited TinyLlama and FLAN-T5 scope checks; these broaden architectural scope modestly but do not alter the paper's main encoder-focused claims.

Additional ablations, evaluation-loss curves (Appendix E), an exploratory score-concentration analysis (Appendix J), and a large-data robustness check at 15,000 training examples (Appendix M) are also reported in the appendix.

Despite the additional frozen probing stage, SAAP's end-to-end runtime is lower than standard LoRA and AdaLoRA in all 20/20 seed-paired comparisons on the four encoder settings (Table 2), making runtime a secondary but practically useful efficiency result.

Ablations (Table 4; Appendix G) confirm that both min–max normalisation and the adaptive Kneedle cutoff contribute materially: removing either component weakens the memory–performance trade-off in the evaluated setting.

Setting	LoRA	AdaLoRA	Sel.	Train	SAAP
BERT / AG News	569.1	898.5	97.9	219.8	**343.0**
BERT / IMDb	1176.3	1372.9	53.0	743.5	**830.1**
RoBERTa / Hate	578.1	961.6	52.4	255.5	**333.0**
DistilBERT / Yelp	479.2	537.6	38.0	369.7	**437.0**

Table 2: Wall-clock runtime (s) on the four main encoder settings, averaged over 5 seeds. **Sel.** denotes SAAP selection time; **Train** denotes SAAP downstream fine-tuning time; **SAAP** is total end-to-end runtime, which includes selection and training plus additional overhead (data loading, evaluation passes, and process initialisation) not separately itemised. This overhead accounts for the difference between the SAAP column and the sum of Sel. and Train.

4 Conclusion

We introduced Sensitivity-Aware Adapter Placement (SAAP), a pre-fine-tuning method for deciding where LoRA adapters should be instantiated under training-memory constraints. Rather than applying adapters uniformly across a predefined module family, SAAP uses a frozen-model sensitivity probe together with normalisation, class-wise

[1] AdaLoRA consistently reports higher final parameter counts than LoRA at the same target rank (e.g. 1.79M vs. 1.19M for BERT-base), reflecting a broader default module set and incomplete rank convergence within 4 epochs.

Model	Dataset	Method	Score ↑	Loss ↓	Avg GPU Mem (MB) ↓	Peak GPU Mem (MB) ↓	Probe Peak GPU (MB)	Params	Density
BERT-base	AG News	Full FT	0.913 ± 0.000	0.291 ± 0.000	1545 ± 14	2819 ± 464	N/A	109.5M	N/A
		LoRA	0.917 ± 0.003	0.277 ± 0.003	587 ± 5	1648 ± 312	N/A	1.19M	N/A
		AdaLoRA	0.895 ± 0.002	0.309 ± 0.001	606 ± 3	1768 ± 245	N/A	1.79M	N/A
		Random	0.861 ± 0.037	0.409 ± 0.091	517 ± 16	959 ± 292	N/A	0.039M	2/73 (2.7%)
		SAAP	**0.884 ± 0.004**	**0.338 ± 0.014**	**482 ± 4**	**545 ± 36**	483	**0.031M**	**2/73 (2.7%)**
BERT-base	IMDb	Full FT	0.919 ± 0.000	0.301 ± 0.000	1685 ± 15	4230 ± 31	N/A	109.5M	N/A
		LoRA	0.911 ± 0.004	0.316 ± 0.014	745 ± 22	3559 ± 85	N/A	1.19M	N/A
		AdaLoRA	0.900 ± 0.001	0.270 ± 0.002	1111 ± 22	3597 ± 37	N/A	1.79M	N/A
		Random	0.854 ± 0.043	0.359 ± 0.088	536 ± 21	1833 ± 522	N/A	0.037M	2/73 (2.7%)
		SAAP	**0.886 ± 0.001**	**0.287 ± 0.003**	**495 ± 5**	**912 ± 29**	483	**0.026M**	**2/73 (2.7%)**
RoBERTa-base	TweetEval Hate	Full FT	0.463 ± 0.000	1.831 ± 0.000	1718 ± 14	2479 ± 69	N/A	124.6M	N/A
		LoRA	0.479 ± 0.008	1.825 ± 0.057	622 ± 3	1158 ± 96	N/A	1.78M	N/A
		AdaLoRA	0.519 ± 0.007	1.058 ± 0.005	650 ± 3	1327 ± 110	N/A	2.38M	N/A
		Random	0.524 ± 0.027	1.080 ± 0.216	575 ± 14	799 ± 98	N/A	0.63M	2/73 (2.7%)
		SAAP	**0.555 ± 0.001**	**0.723 ± 0.001**	**548 ± 0**	**587 ± 16**	543	**0.63M**	**2/73 (2.7%)**
DistilBERT	Yelp Polarity	Full FT	0.943 ± 0.000	0.218 ± 0.000	1128 ± 20	3622 ± 25	N/A	66.9M	N/A
		LoRA	0.934 ± 0.002	0.199 ± 0.002	506 ± 10	2557 ± 84	N/A	0.81M	N/A
		AdaLoRA	0.913 ± 0.003	0.213 ± 0.002	613 ± 23	2591 ± 31	N/A	0.92M	N/A
		Random	0.929 ± 0.005	0.193 ± 0.009	426 ± 33	1926 ± 334	N/A	0.70M	4.6/36 (12.8%)
		SAAP	**0.926 ± 0.002**	**0.201 ± 0.004**	**367 ± 12**	**1251 ± 34**	323	**0.70M**	**4.6/36 (12.8%)**

Table 1: Mean ± standard deviation over 5 seeds. Score denotes Accuracy except for TweetEval Hate, where Macro-F1 is reported. *Peak GPU Mem* is peak GPU memory allocated during downstream fine-tuning only. *Probe Peak GPU* reports the peak GPU memory allocated during the SAAP selection stage (frozen-model forward/backward pass); this stage is absent for other methods (N/A). Probe peak is identical across all 5 seeds (std $= 0$) and is therefore reported as a single value. The true worst-case peak for SAAP is the higher of the two figures; in all four settings the probe peak is lower than or comparable to the fine-tuning peak. Density (Random and SAAP only) reports mean selected module instances out of the total candidate set $|\mathcal{M}|$; Full FT, LoRA, and AdaLoRA do not perform instance-level selection. Where Random and SAAP share the same density (e.g. 2/73 for BERT-base settings), their parameter counts may still differ because the two methods select different modules, and different linear projections within the same architecture can have different output dimensions and therefore different LoRA parameter counts. Entries reported as 0.000 indicate standard deviations that rounded to zero at the displayed precision.

ranking, adaptive cutoff selection, and stability filtering to retain only a sparse set of task-relevant modules before downstream fine-tuning.

Across four encoder-based sequence-classification settings, the main empirical result is not uniform accuracy improvement, but a more favourable memory–performance trade-off. Compared with standard LoRA and AdaLoRA, SAAP uses substantially fewer trainable parameters and often reduces both average and peak GPU memory while maintaining competitive predictive performance. Comparisons with a density-matched random sparse-placement baseline further suggest that these gains arise from informed placement rather than sparsity alone. Despite the additional probing stage, SAAP also reduces end-to-end runtime on the main encoder settings, and the constrained-memory feasibility check indicates that it remains trainable in software-simulated low-VRAM conditions where the denser baselines do not.

More broadly, these findings suggest that adapter placement is an important degree of freedom in memory-aware PEFT, alongside parameterisation and rank allocation. The scope of the present evidence is intentionally limited: the main experiments focus on encoder-based sequence classification, and the appendix provides only preliminary scope-broadening checks on TinyLlama-1.1B and FLAN-T5-small. Extending the evaluation to broader task families, larger data regimes, and stronger lightweight adaptation baselines remains an important direction for future work.

5 Limitations

Empirical and architectural scope. All main claims are restricted to encoder-based sequence classification on small subsets (5,000 training and 1,000 evaluation examples, 4 tasks, 3 model families). The appendix reports limited supplementary checks on TinyLlama-1.1B and FLAN-T5-small, but these are not sufficient to establish generalisation to decoder-only LLMs at other scales, encoder–decoder models, multilingual settings, or generative tasks. Broader evaluation across task types and data regimes remains future work.

Frozen-probe assumption. SAAP computes sensitivity scores on a fully frozen model as a proxy for fine-tuning sensitivity. There is no theoretical guarantee that frozen-model gradients correlate reliably with gradients during fine-tuning, and this assumption is not directly validated beyond the observation that SAAP outperforms budget-matched random placement. Settings where the frozen signal is uninformative, such as heavily domain-shifted data or architectures with pronounced gradient vanishing, may not be well served by the probing stage.

Score and pipeline design. The GWM score is a practical hybrid of Taylor- and Fisher-style reasoning rather than a criterion derived from either framework alone, and is not an optimal estimator of module utility. The pipeline hyperparameters (probe pool size, stability passes N, retention threshold) were fixed by preliminary checks on the same datasets used for evaluation, so their optimality for unseen tasks or architectures is not established. Ablations are further restricted to a single setting (BERT-base/IMDb), which limits conclusions about individual pipeline components.

Baseline comparison scope. We compare against full fine-tuning, standard LoRA, AdaLoRA, and a budget-matched random baseline. The random baseline is given the same module budget as SAAP post-hoc rather than selecting its own budget independently, which isolates the effect of informed placement but is inherently favourable to SAAP. Results should be read as evidence that SAAP is competitive within this comparison set, not as a claim of uniform superiority over all lightweight adaptation methods.

Memory reporting and statistical evaluation. Reported GPU memory covers downstream fine-tuning only; the probing stage is excluded, so SAAP's true peak may be modestly higher than the figures shown. Additionally, no per-component allocator-level decomposition is provided; the figures are end-user PyTorch allocated memory measurements rather than a controlled attribution of each megabyte to weights, gradients, optimiser state, or activations.

The constrained-memory experiment is a software-simulated feasibility check, not a real-device benchmark. Results are averaged over 5 seeds with no formal significance testing; the score gap between SAAP and random placement on DistilBERT/Yelp (0.003) is smaller than the standard deviation of either method, and all memory figures are hardware-specific.

6 Ethical Considerations

This work contributes to the literature on parameter-efficient fine-tuning by studying how to reduce the training-time memory cost of adapting pretrained language models. By instantiating adapters only in task-sensitive modules, SAAP reduces the number of trainable parameters relative to uniform LoRA baselines and often lowers GPU memory usage during training. These reductions may also lower computational overhead in practice, although we do not directly measure runtime energy use or carbon emissions.

SAAP may also improve transparency in model adaptation, since the selected adapter locations are explicit and inspectable, making the adaptation process easier to analyse than uniformly applied PEFT methods. That said, interpretability should not be overstated: identifying which modules were selected does not by itself explain model behaviour or guarantee fairness.

We also acknowledge dataset-related risks. One of our benchmarks, TweetEval Hate, involves subjective annotations and potentially socially sensitive language. Models adapted on this dataset may inherit annotation artefacts or biases. We therefore recommend fairness and bias evaluation before deploying SAAP-adapted systems in high-stakes or moderation-related applications.

Finally, large language models (ChatGPT and Gemini) were used to assist with improving the clarity and presentation of the writing. They were not used to generate experimental results, run analyses, or determine the scientific conclusions of the paper.

References

Frederik Benzing. 2022. Unifying importance based regularisation methods for continual learning. In *Proceedings of the 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 2372–2396. PMLR.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 10088–10115.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea

Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning (ICML)*, pages 2790–2799.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Liyang Liu, Shilong Zhang, Zhanghui Kuang, Aojun Zhou, Jing-Hao Xue, Xinjiang Wang, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. 2021. Group fisher pruning for practical network compression. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 7021–7032. PMLR.

Pavlo Molchanov, Arun Mallya, Jan Kautz, Insung Ihm, and Xia Shen. 2019. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11264–11272.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2017. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations*.

Aleksandra I. Nowak, Otniel-Bogdan Mercea, Anurag Arnab, Jonas Pfeiffer, Yann Dauphin, and Utku Evci. 2024. Towards optimal adapter placement for efficient transfer learning. *arXiv preprint arXiv:2410.15858*.

Victor Sanh, Thomas Wolf, and Alexander M. Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 20378–20389.

Ville Satop"a"a, Jeannie Albrecht, David Irwin, and Barath Raghavan. 2011. Finding a 'kneedle' in a haystack: Detecting knee points in system behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pages 166–171. IEEE.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Qingru Zhang, Min Chen, Alexander Bukharin, Penn He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

A Full Selection Procedure

For completeness, we summarise the full stability-aware adapter-selection pipeline used by SAAP. The procedure operates before downstream fine-tuning and determines the subset of candidate modules that will receive LoRA adapters.

1. **Candidate restriction.** Define the candidate module set $\mathcal{M}$ as the standard LoRA-style linear projections considered for the corresponding architecture family.

2. **Fixed class-balanced probe pool.** Construct a small class-balanced probe pool from the training data once per run. The backbone model remains frozen throughout this stage.

3. **Stability-pass subsampling.** For each stability pass, draw a class-balanced subsample from this fixed probe pool rather than sampling a new subset from the full training data. In the default configuration, this subsampling is performed without replacement within each pass (i.e., fold-style subsampling). Keeping the probe source constant across passes preserves class balance and improves reproducibility, while repeated subsampling tests whether the same modules remain salient under small perturbations of the probing evidence.

4. **Scoring.** Run a forward and backward pass on the frozen model and compute the activation-weighted squared-gradient score for each candidate module m:

$$S_{\mathrm{GWM}}^{(c)}(m) = \sum |a \odot g^2|.$$

5. **Per-layer normalisation.** Group candidate modules by Transformer block and apply min–max normalisation within each group.

6. **Class-wise ranking and cutoff selection.** For each class-specific score list, rank modules by normalised score in descending order. Apply the Kneedle algorithm (Satopaa et al., 2011), which detects the point of maximum perpendicular distance from the chord connecting the first and last points of the score-rank curve, to

determine an adaptive cutoff kc. This yields a data-driven budget rather than a fixed global Top-K threshold. Class-specific rankings are used to preserve modules that may be especially relevant to one label class but would be suppressed under a single pooled ranking.

7. **Aggregation across classes.** Form the union of selected modules across all class-specific rankings. This preserves coverage of class-specific salience patterns, though it can reduce sparsity when the number of classes is large.

8. **Stability filtering.** Repeat Steps 3–7 for $N = 20$ passes and retain only modules selected in at least 99% of passes, which with $N = 20$ corresponds to being selected in every pass. We adopt this strict criterion to favour reproducibility of the retained set; preliminary checks suggested that lower thresholds admitted more variable module selections. The choice of $N = 20$ is a practical compromise between stability estimation and probing cost.

9. **Adapter instantiation.** Insert LoRA adapters only into the retained module set and proceed with downstream fine-tuning.

B Exact Random Seeds and Probe Configuration

Across all repeated experiments, we use data/selection seeds $\{45, 46, 47, 48, 49\}$, while keeping the model initialisation seed fixed at 42 for all runs. In each run, SAAP constructs a fixed class-balanced probe pool by sampling 50 training examples per label class from the training split. This yields 100 total probe examples for binary tasks and 200 total probe examples for 4-way AG News. Stability selection then performs $N = 20$ passes; in each pass, it draws a class-balanced subsample consisting of half of the probe pool per class (25 per class in the default configuration), sampled without replacement within the pass.

For each stability pass, a Kneedle-style knee cutoff is applied independently to the descending per-class (per-layer normalised) score curves, and the selected module sets are unioned across classes to produce a per-pass selection. We repeat this process for $N = 20$ passes and retain only modules selected in at least 99% of passes, which in this setting corresponds to being selected in every pass. We use this strict criterion as a conservative stability filter: preliminary checks suggested that looser thresholds yielded more variable retained module sets across passes.

C Probe-Pool Size Selection

Because SAAP introduces a probing stage before fine-tuning, we conducted a preliminary pilot sensitivity analysis over probe-pool sizes ranging from 10 to 5,000 total probe examples to choose a practical default. We varied the total number of probe examples and tracked downstream *validation* performance in this pilot setting only. Applying Kneedle to the resulting probe-size performance curve identified a practical knee at approximately 100 total probe examples in the binary pilot setting. We therefore fix the probe budget to 50 examples per class in all main experiments, rather than tuning it separately for each dataset or model.

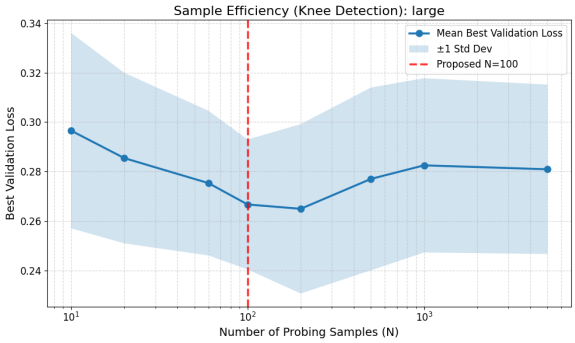

Figure 1: Pilot sensitivity analysis for probe-pool size selection. Performance improves rapidly at small probe sizes and then plateaus; Kneedle identifies a practical knee around 100 total probe examples in the binary pilot setting (50 per class), which is used as the fixed default in the main experiments.

D Selection of Importance Heuristic

Our heuristic comparison is empirical, but the candidate signals are not theoretically unrelated. Activation-only scores measure engagement without task sensitivity, while gradient-only scores measure local loss sensitivity without regard to whether the corresponding representation is strongly expressed. Hybrid activation–gradient scores can be interpreted as local loss-perturbation proxies under gating-style Taylor approximations, and squared-gradient variants additionally connect to diagonal Fisher-style importance measures (Molchanov et al., 2017, 2019; Liu et al., 2021; Benzing, 2022). We therefore view the heuristic search space as a family of increasingly task-aware local saliency

proxies rather than as unrelated scoring rules.

To determine a suitable sensitivity signal for SAAP, we compared activation-only, gradient-only, and combined gradient–activation heuristics. In the evaluated setting, hybrid signals generally outperformed single-signal alternatives, and the activation-weighted squared-gradient formulation ($a \cdot g^2$) provided the strongest empirical balance of performance and stability.

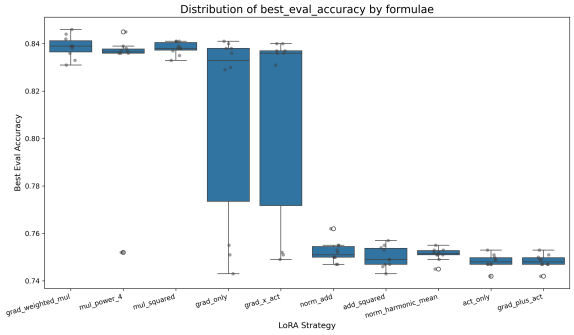

Figure 2: Comparison of candidate importance heuristics. Hybrid gradient–activation methods outperform single-signal alternatives in the evaluated setting.

E Training Stability and Evaluation Loss

Figure 3 compares evaluation-loss trajectories for SAAP and the budget-matched random baseline on representative AG News and IMDb settings. In BERT-base/AG News, BERT-base/IMDb, and FLAN-T5-small/AG News, SAAP reaches a lower final loss and exhibits less erratic behaviour than random sparse placement. The difference is especially pronounced for FLAN-T5-small, where random placement remains substantially higher-loss throughout training. In TinyLlama AG News, however, the two methods are considerably closer, suggesting that the optimisation advantage of informed placement is setting-dependent. These plots should be read as descriptive evidence rather than as a formal convergence guarantee.

F Seed-Level Paired Robustness

To complement the mean ± standard deviation summaries in the main paper, Table 3 reports seed-paired comparisons on the four main encoder settings. For each setting and seed, we compare SAAP directly against the budget-matched random baseline on score and peak GPU memory, and against standard LoRA and AdaLoRA on total end-to-end runtime. These results should be interpreted as descriptive robustness evidence rather than as formal significance testing.

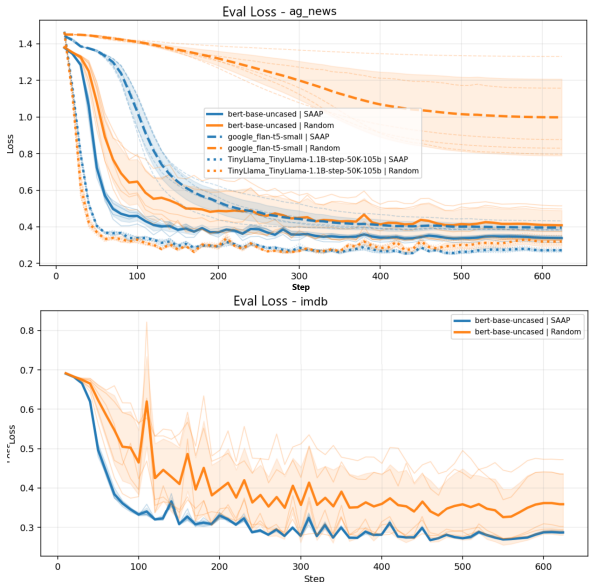

Figure 3: Evaluation-loss trajectories for SAAP and the budget-matched random baseline on representative AG News and IMDb settings. Shaded regions denote standard deviation across seeds. SAAP generally shows lower-loss and more stable behaviour in the BERT-base and FLAN-T5-small settings, while the TinyLlama AG News trajectories are more similar.

Table 3 reinforces the main empirical picture. The predictive comparison against random placement remains task-dependent, but the memory and runtime patterns are considerably more consistent: SAAP uses lower peak GPU memory than the random baseline in 19/20 seed-paired comparisons, and its end-to-end runtime is lower than both standard LoRA and AdaLoRA in all 20/20 seed-paired comparisons on the four main encoder settings.

G Selection Pipeline Ablations

To test whether the main components of the SAAP selection pipeline contribute in practice, we conduct an ablation study on BERT-base/IMDb. We evaluate the effect of removing the per-layer min–max normalisation step and of replacing the adaptive Kneedle cutoff with a fixed global Top-K strategy. These ablations are intended to test whether SAAP's behaviour depends only on sparse adapter placement, or whether the normalisation and adaptive cutoff components contribute materially.

As shown in Table 4, both components contribute in the evaluated setting. Removing min–max normalisation lowers accuracy from 0.886 to 0.850 at the same adapter budget, consistent with per-layer normalisation mitigating depth-related score drift. Replacing the adaptive Kneedle cutoff with a fixed Top-K rule lowers accuracy to 0.865

Setting	> Rand Score	< Rand Peak	< LoRA Time	< Ada Time
BERT / AG	3/5	5/5	5/5	5/5
BERT / IMDb	4/5	4/5	5/5	5/5
RoBERTa / Hate	4/5	5/5	5/5	5/5
Distil / Yelp	2/5	5/5	5/5	5/5
Total	**13/20**	**19/20**	**20/20**	**20/20**

Table 3: Seed-paired robustness comparisons on the four main encoder settings. Score is Accuracy except for RoBERTa/TweetEval Hate, where Macro-F1 is used. "> Rand Score" counts seeds in which SAAP exceeds the budget-matched random baseline in score. "< Rand Peak" counts seeds in which SAAP uses less peak GPU memory than the random baseline. Time columns count seeds in which SAAP has lower total end-to-end wall-clock runtime.

Method	Score ↑	Avg GPU ↓	Params
SAAP	**0.886**	495 MB	**0.026M**
w/o Min–Max Norm.	0.850	**367 MB**	0.026M
Fixed Top-K	0.865	442 MB	0.063M

Table 4: Selection-pipeline ablations on BERT-base (IMDb), averaged over 5 seeds. Removing min–max normalisation reduces predictive performance, while replacing the adaptive Kneedle cutoff with a fixed Top-K rule both lowers accuracy and increases parameter count.

while increasing the parameter count from 0.026M to 0.063M and raising peak memory. Notably, the fixed Top-K ablation performs worse than SAAP despite using more than twice as many trainable parameters, which strengthens the case that SAAP's gains are not explained by sparsity alone. Together, these results indicate that SAAP's behaviour is not reducible to sparse adapter placement, and support the claim that both the normalisation and adaptive cutoff stages contribute materially to the final memory–performance trade-off.

While Table 4 isolates only min–max normalisation and adaptive cutoff selection, the same design logic also motivates the remaining pipeline components. We use the class-wise union to avoid suppressing modules strongly salient for only part of the label space, and we use a 99% retention threshold over $N = 20$ passes as a conservative filter against one-off probe noise. These choices are currently supported by preliminary stability checks rather than a full hyperparameter sweep, and should be read as empirically motivated defaults rather than definitive optimal settings.

H Preliminary Evidence on Architectural Generality

To probe whether the placement principle extends beyond encoder architectures, we report results

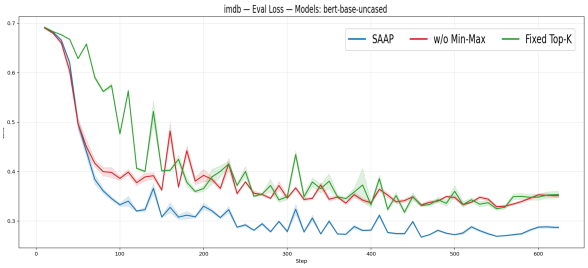

Figure 4: Evaluation loss trajectories for the selection-pipeline ablations on BERT-base (IMDb).

on a decoder-only model (TinyLlama-1.1B) and an encoder–decoder model (FLAN-T5-small) on AG News and TweetEval Hate. These results are reported as supporting appendix evidence only and should not be interpreted as comprehensive cross-architecture validation; broader evaluation across model scales, tasks, and training regimes remains future work.

Table 5 summarises performance. To ground the results, we examine which modules SAAP selected in each setting, as these patterns provide interpretable evidence that the sensitivity probe responds to task structure rather than selecting arbitrarily.

FLAN-T5-small. In the FLAN-T5-small settings, SAAP retains a very sparse adapter set whose highest-scoring modules are concentrated around the classification head and decoder cross-attention. On TweetEval Hate, the score profile is sharply concentrated, yielding a clear knee and a small stable retained set. On AG News, the retained set remains sparse but broadens slightly through the class-wise union across four label classes. Across passes, some additional modules appear transiently in individual class-specific rankings but do not survive the 99% stability filter, suggesting their apparent salience is not reproducible under small perturbations of the probe subsample. The modules that survive stability filtering are dominated by decoder cross-attention and task-head components, which is architecturally plausible for an encoder–decoder classifier: cross-attention is the mechanism through which encoder representations are routed into the decoder for label prediction, while the classification head is directly tied to the downstream loss. Empirically, SAAP substantially outperforms the density-matched random baseline on AG News (0.864 vs. 0.634) and remains close to it on TweetEval Hate (0.450 vs. 0.443), while using less peak GPU memory in both cases. The large AG News gap suggests that uninformed sparse placement

may be especially unreliable in encoder–decoder architectures, where randomly omitting adapters can disrupt the cross-attention pathways most relevant to classification.

TinyLlama-1.1B. The TinyLlama settings exhibit a different pattern. Relative to the encoder and FLAN-T5 results, SAAP retains a broader sparse adapter set, with selected modules concentrated mainly in value projections, output projections, and feed-forward down-projection layers in earlier-to-middle blocks. The score distribution is much flatter than in the FLAN-T5 or encoder settings, with no comparably sharp knee: scores decline gradually rather than collapsing after a small number of dominant modules. This flatter profile is architecturally plausible in a decoder-only model, where task sensitivity appears to be distributed across a wider portion of the network rather than concentrated in a structurally distinct task head. The consistent preference for value and output projections, and for feed-forward down-projections, over query and key projections is also interpretable under the GWM score, as these modules more directly govern the content propagated forward to support classification. Empirically, SAAP modestly outperforms random placement in score on both datasets, but its peak memory is slightly higher than the random baseline in both cases. This is consistent with the flatter score distribution: when sensitivity is less concentrated, the adaptive cutoff retains a broader set of modules, which weakens the memory advantage that SAAP achieves more clearly in settings with sharper score concentration.

Taken together, these results suggest that the placement principle is not obviously encoder-exclusive. Across both non-encoder families,

sensitivity-guided selection yields interpretable, architecture-dependent module patterns and outperforms uninformed sparse placement in score on 3 of the 4 reported settings; on TinyLlama/TweetEval Hate the gap of 0.005 is smaller than the standard deviation of either method and should not be read as a reliable difference.

The memory advantage is clearest for FLAN-T5, where score concentration permits aggressive sparsity, and weaker for TinyLlama, where the flatter score profile limits how sparse the retained set can become. These appendix results should nevertheless be interpreted cautiously: they broaden architectural scope, but wider validation across decoder-only scales, generation tasks, and larger training regimes remains necessary before strong cross-architecture claims can be made.

I Constrained-Memory Feasibility Check

To test behaviour under a strict memory budget, we simulate a constrained profile with 2 CPU threads and a 1.25 GB VRAM cap. Table 6 shows that in two representative encoder settings, SAAP and the random baseline complete training while full fine-tuning, LoRA, and AdaLoRA do not; SAAP achieves the lower peak footprint among the methods that succeed. These results should be interpreted narrowly as evidence of trainability under a software-imposed memory limit, not as a substitute for real-device benchmarking.

Setting	Method	Status	Peak VRAM
BERT / AG News Batch = 8	Full FT	*Failed*	OOM ($>$ 1.25 GB)
	LoRA	*Failed*	OOM ($>$ 1.25 GB)
	AdaLoRA	*Failed*	OOM ($>$ 1.25 GB)
	Random	Success	746 MB
	SAAP	**Success**	**560 MB**
DistilBERT / Yelp Batch = 4	Full FT	*Failed*	OOM ($>$ 1.25 GB)
	LoRA	*Failed*	OOM ($>$ 1.25 GB)
	AdaLoRA	*Failed*	OOM ($>$ 1.25 GB)
	Random	Success	937 MB
	SAAP	**Success**	**765 MB**

Table 6: Software-simulated constrained-memory experiment under a 1.25 GB VRAM cap. Only SAAP and Random completed training; SAAP used lower peak VRAM in both settings. Results reflect feasibility under a software-imposed memory limit, not real-device throughput or deployment readiness.

J Module Selection Analysis

Exploratory analysis of score concentration. As an exploratory analysis, we examine whether the *score concentration* of the selected modules (the fraction of total probe-stage sensitivity mass captured by the finally retained set) is associated with the downstream score gap relative to standard LoRA. For each setting, we compute from the probing-stage logs the ratio of the summed mean module score of the selected modules to the summed mean score of all candidate modules, averaged over all available seeds. Table 7 and Figure 5 report the result across the four main encoder settings.

Across the four main encoder settings, higher score concentration is associated with a smaller downstream score gap relative to standard LoRA. Intuitively, when the retained modules capture a large fraction of the probe-stage sensitivity mass, the modules left without adapters may contribute relatively less task-relevant signal, so sparse place-

Setting	Method	Score ↑	Loss ↓	Avg GPU Mem (MB) ↓	Peak GPU Mem (MB) ↓	Probe Peak GPU (MB)	Params	Density
TinyLlama 1.1B AG News	Full FT	OOM	OOM	13456 ± 88	>19402	N/A	1.10B	N/A
	LoRA	0.905 ± 0.002	0.348 ± 0.020	4551 ± 5	12085 ± 649	N/A	1.86M	N/A
	AdaLoRA	0.907 ± 0.002	0.268 ± 0.003	4653 ± 11	12171 ± 715	N/A	2.78M	N/A
	Random	0.908 ± 0.004	0.320 ± 0.027	4406 ± 30	10179 ± 702	N/A	0.78M	16/154 (10.6%)
	SAAP	0.911 ± 0.006	0.272 ± 0.008	4407 ± 10	10456 ± 675	4012	0.53M	16/154 (10.6%)
TinyLlama 1.1B TweetEval Hate	Full FT	0.461 ± 0.000	4.108 ± 0.000	13708 ± 22	19453 ± 329	N/A	1.03B	N/A
	LoRA	0.510 ± 0.011	3.114 ± 0.248	4339 ± 14	6948 ± 320	N/A	1.85M	N/A
	AdaLoRA	0.509 ± 0.005	1.072 ± 0.006	4418 ± 12	7351 ± 197	N/A	2.78M	N/A
	Random	0.501 ± 0.012	2.110 ± 0.456	4208 ± 15	6203 ± 241	N/A	0.441M	9.4/154 (6.1%)
	SAAP	0.506 ± 0.010	1.081 ± 0.036	4233 ± 9	6327 ± 319	4011	0.212M	9.4/154 (6.1%)
FLAN-T5-small AG News	Full FT	0.858 ± 0.000	0.441 ± 0.000	978 ± 7	3226 ± 622	N/A	60.7M	N/A
	LoRA	0.878 ± 0.002	0.372 ± 0.013	430 ± 5	1885 ± 244	N/A	0.35M	N/A
	AdaLoRA	0.406 ± 0.021	1.357 ± 0.004	440 ± 3	2242 ± 505	N/A	0.53M	N/A
	Random	0.634 ± 0.178	0.998 ± 0.234	401 ± 7	1832 ± 388	N/A	0.035M	4/96 (4.0%)
	SAAP	0.864 ± 0.004	0.396 ± 0.021	344 ± 2	642 ± 57	296	0.025M	4/96 (4.0%)
FLAN-T5-small TweetEval Hate	Full FT	0.452 ± 0.000	0.818 ± 0.000	927 ± 6	1885 ± 154	N/A	60.8M	N/A
	LoRA	0.436 ± 0.012	1.221 ± 0.043	388 ± 4	1110 ± 72	N/A	0.352M	N/A
	AdaLoRA	0.385 ± 0.002	0.686 ± 0.000	397 ± 1	1190 ± 71	N/A	0.529M	N/A
	Random	0.443 ± 0.059	0.698 ± 0.014	353 ± 23	716 ± 163	N/A	0.019M	2/96 (2.1%)
	SAAP	0.450 ± 0.005	0.789 ± 0.004	326 ± 0	548 ± 21	296	0.012M	2/96 (2.1%)

Table 5: Limited non-encoder scope checks. Score is Accuracy for AG News and Macro-F1 for TweetEval Hate. Memory is reported as average and peak GPU memory allocated during downstream fine-tuning only. *Probe Peak GPU* reports the peak GPU memory allocated during the SAAP selection stage (frozen-model forward/backward pass); this stage is absent for other methods (N/A). Probe peak standard deviation is $\leq$1 MB across all 5 seeds and is omitted for brevity. For TinyLlama 1.1B / AG News, Full FT completed forward passes (memory logged) but did not converge to a valid score or loss (OOM during training). These results are reported as supporting appendix evidence only and should not be interpreted as comprehensive cross-architecture validation. Entries reported as 0.000 indicate standard deviations that rounded to zero at the displayed precision.

Setting	Seeds	Conc.	Gap vs. LoRA
BERT / AG	5	31.0 ± 0.9%	−0.033
BERT / IMDb	5	61.3 ± 0.4%	−0.025
Distil / Yelp	5	79.7 ± 0.1%	−0.008
RoBERTa / Hate	5	99.0 ± 0.0%	+0.076

Table 7: Score concentration of SAAP-selected modules (mean ± standard deviation across seeds) and the corresponding score gap relative to standard LoRA (positive = SAAP outperforms LoRA). Score denotes Accuracy except for RoBERTa/TweetEval Hate, where Macro-F1 is used. Score concentration is computed from the probing stage before downstream fine-tuning, as the fraction of total sensitivity mass captured by the retained adapter set.

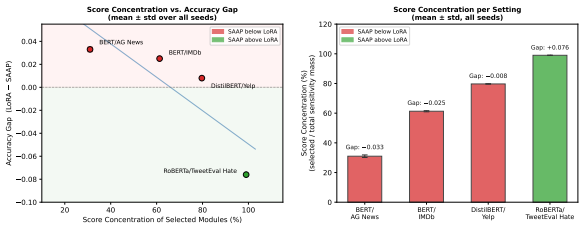

Figure 5: Score concentration of selected modules versus score gap relative to LoRA (*left*), and per-setting summary with error bars (*right*), across the four main encoder settings. Error bars denote ±1 standard deviation across seeds for the concentration metric. Higher score concentration is associated with a smaller downstream score gap relative to LoRA in these settings. This pattern is descriptive and should not be interpreted as a validated predictive law.

ment incurs a smaller performance cost. Conversely, lower concentration may indicate that substantial sensitivity mass remains outside the retained set, in which case the score gap relative to uniform LoRA may be larger.

The concentration metric is also fairly stable across seeds, with standard deviations ranging from 0.0% to 0.9% across the four settings. Because this ratio is computable from the probing stage alone, which adds only 38–98 seconds of wall-clock time (Table 2) before downstream fine-tuning, it may serve as a useful descriptive diagnostic prior to full training. We emphasise, however, that this analysis is exploratory, based on only four encoder settings, and should not be interpreted as establishing a universal predictive threshold for adapter quality.

K Selection Stability Across Seeds

Because SAAP relies on a pre-fine-tuning module-selection stage, we also summarise how stable the selected adapter sets are across random seeds on the four main encoder settings. Table 8 reports the mean number of selected modules, the range across seeds, the number of unique selected-module patterns observed over 5 seeds, and the mean pairwise Jaccard overlap between selected sets.

Setting	Mean Sel.	Range	Unique	Mean Jaccard
BERT / AG	2.0	2–2	2	0.73
BERT / IMDb	2.0	2–2	1	1.00
RoBERTa / Hate	3.0	3–3	1	1.00
Distil / Yelp	4.6	4–5	2	0.88

Table 8: Selection stability of SAAP across 5 seeds on the four main encoder settings. "Mean Sel." is the mean number of selected modules. "Unique" counts distinct selected-module patterns observed across seeds. "Mean Jaccard" is the mean pairwise Jaccard overlap of the selected module sets.

The selected module sets are generally highly stable. BERT/IMDb and RoBERTa/TweetEval Hate each produce a single selection pattern across all five seeds with mean pairwise Jaccard overlap of 1.00, while DistilBERT/Yelp remains highly consistent across seeds despite alternating between 4 and 5 selected modules. BERT/AG News shows somewhat lower stability than the other encoder settings, but even there the selected module count is fixed and only two distinct patterns appear across five seeds.

L Accuracy–Memory Pareto Frontier

Figure 6 visualises the score/peak-memory trade-off across all eight reported settings. Each subplot shows mean performance over seeds; the Pareto frontier line connects methods that are not dominated on both axes simultaneously.

Across the four main encoder settings, SAAP occupies the low-memory end of the frontier in every case. On BERT-base/AG News and BERT-base/IMDb, SAAP anchors the bottom-left of a frontier that runs through standard LoRA and Full FT at progressively higher memory and accuracy. On RoBERTa-base/TweetEval Hate, SAAP achieves both the highest score and the lowest peak memory among all methods, dominating the frontier outright. DistilBERT/Yelp Polarity is the exception: the drawn frontier runs through Random, LoRA, and Full FT, while SAAP sits slightly below the frontier in score (0.926 vs. Random's 0.929) at substantially lower memory (1,251 vs. 1,926 MB). Since no method simultaneously achieves higher score *and* lower memory than SAAP, it remains a Pareto-efficient operating point, but it does not anchor the frontier in this setting. For the non-encoder scope checks, the pattern is more varied. In both FLAN-T5-small settings, SAAP occupies the lowest-memory position by a large margin and lies on the Pareto frontier. In the TinyLlama-1.1B settings, the methods cluster more tightly in memory, and no clean frontier emerges: SAAP achieves competitive or slightly higher score than the other methods but does not achieve a clear memory advantage, consistent with the flatter score distribution observed in decoder-only models (Appendix H).

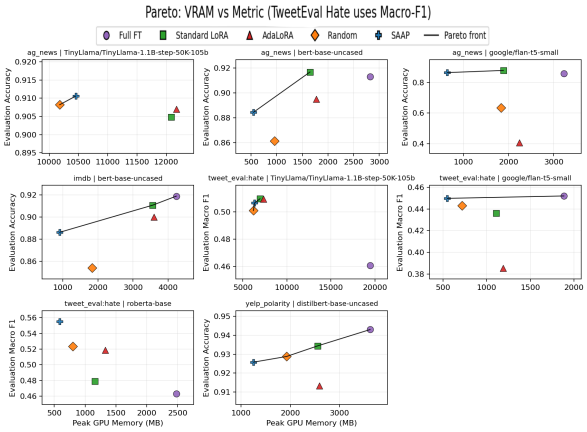

Figure 6: Score (Accuracy or Macro-F1 for TweetEval Hate) vs. peak GPU memory allocated during training, across all eight reported settings. Each point is the mean over 5 seeds. The Pareto frontier line connects methods that are not dominated on both axes simultaneously. SAAP anchors the low-memory end of the frontier in the four main encoder settings and in both FLAN-T5-small settings. In DistilBERT/Yelp Polarity, SAAP is a Pareto-efficient point but does not anchor the frontier (see Appendix G and the main text). In the TinyLlama-1.1B settings no clean frontier is drawn, reflecting the tighter clustering of methods in that regime.

AdaLoRA performs notably poorly on FLAN-T5-small/AG News (0.406 accuracy), appearing as an outlier in the bottom-right of that subplot; this is discussed in Appendix H.

M Large-Data Robustness Check

The main experiments use 5,000 training examples, reflecting a memory-constrained adaptation scenario but limiting claims about behaviour at larger data scales. As a preliminary robustness check, we re-ran all five methods on BERT-base/IMDb with 15,000 training and 5,000 evaluation examples, 4 epochs, and 3 seeds ($\{45, 46, 47\}$), keeping all other hyperparameters identical to the main experiments. Because these runs were conducted on different hardware, GPU memory figures are not reported here and cannot be compared to Table 1. These results should be interpreted as supporting robustness evidence only.

Method	Accuracy ↑	Eval Loss ↓
Full FT	0.935 ± 0.000	0.289 ± 0.000
LoRA	0.937 ± 0.001	0.237 ± 0.004
AdaLoRA	0.928 ± 0.001	0.194 ± 0.001
Random	0.900 ± 0.036	0.252 ± 0.061
SAAP	**0.920 ± 0.000**	**0.218 ± 0.002**

Table 9: Large-data robustness check: BERT-base/IMDb, 15,000 training examples, 3 seeds. GPU memory is not reported as these runs used different hardware than the main experiments. Score is accuracy; eval loss is best checkpoint loss on the 5,000-example evaluation set.

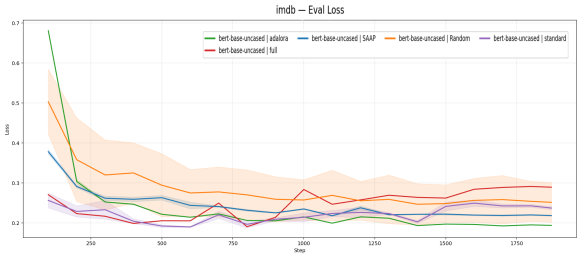

Figure 7: Evaluation loss during training on the large-data setting (BERT-base/IMDb, 15,000 training examples). Shaded regions denote ±1 standard deviation across 3 seeds. Full FT shows a gradual rise in eval loss after initial convergence, while SAAP and standard LoRA remain stable.

The main patterns hold at this scale. The accuracy gap between SAAP and LoRA narrows slightly (-0.018 vs. -0.025 in the main experiments), consistent with parameter-richer methods benefiting more from additional data. Random placement retains high variance (std $= 0.036$), confirming that uninformed sparse placement remains unstable at larger data scale. SAAP selects identical modules across all three seeds, focusing on the attention output projection in layers 9 and 11, matching the small-data selection exactly. Figure 7 shows evaluation loss trajectories during training; Full FT exhibits a characteristic rise in eval loss after initial convergence, while SAAP and LoRA remain stable throughout.