SEESAW : BUDGET-PRESERVING RANK REALLOCATION FOR LOW-RANK ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Conventional LoRA assigns every Transformer layer the same rank, implicitly assuming uniform adaptation demand across layers. We instead cast layer-wise rank selection as a fixed-budget *allocation* problem and introduce Seesaw, which redistributes a constant global rank budget across layers using an exponentially-smoothed, rank-normalized gradient signal. At each allocation step, Seesaw identifies the highest-scoring receiver and lowest-scoring donor layer and, when their relative importance gap exceeds a threshold τ, transfers a single rank unit between them—resizing the corresponding A/B matrices and resetting optimizer state while leaving the total rank sum unchanged. On six NLU benchmarks with BERT-base, Seesaw outperforms uniform-rank LoRA, AdaLoRA, and DyLoRA at matched budgets; controlled ablations against fixed non-uniform and importance-inverted allocation directions show the gains stem specifically from correctly-directed reallocation, not non-uniformity alone. We further perform against standard LoRA on DeBERTa-v1, and benchmark against BitFit, Houlsby/Pfeiffer Adapters, LoRA, and AdaLoRA on DeBERTa-v3 under identical parameter budgets. Our results show that *where* a fixed adaptation budget is placed matters as much as *how large* it is, and that a lightweight first-order signal is enough to discover effective, architecture-dependent placements without second-order or SVD-based machinery.

1 INTRODUCTION

Parameter-efficient fine-tuning (PEFT) has become the default approach for adapting large pretrained language models to downstream tasks while avoiding the cost of updating every parameter. Among PEFT methods, Low-Rank Adaptation (LoRA) (Hu et al., 2022) is particularly effective: it freezes the pretrained backbone and represents task-specific updates as a pair of low-rank matrices, substantially reducing the number of trainable parameters while retaining competitive downstream performance. In its standard form, however, LoRA assigns an *identical* rank to every adapted layer. This single design choice simultaneously fixes two distinct quantities: the total adaptation capacity available to the model, and how that capacity is distributed across layers.

The uniform-rank assumption is convenient, but it is not obviously correct. Different downstream objectives draw on different representations within a pretrained Transformer, and there is no reason to expect every layer to require the same adaptation capacity for every task. Addressing this by increasing the rank uniformly enlarges the total parameter budget rather than correcting its distribution; addressing it by hand-tuning per-layer ranks introduces a search problem that scales with the number of layers and must be redone for every new task. Both routes conflate a question about *quantity* (how much capacity to add) with a separate question about *placement* (where that capacity should go). This work isolates the second question: given a fixed global rank budget, how should it be distributed across layers, and can this distribution be discovered automatically and cheaply during training?

Existing adaptive-rank methods relax the uniform-rank assumption but do not directly target this placement question under a fixed budget. AdaLoRA (Zhang et al., 2023b) allocates parameter budget according to importance estimates derived from an SVD-based parameterization, progressively pruning less important singular components; this changes the *effective* rank of each layer but operates at the granularity of singular values and requires repeated decomposition during training.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

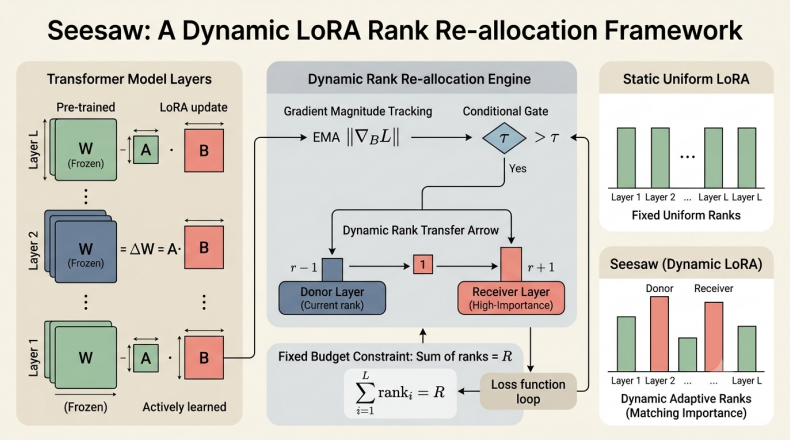


Figure 1: **Overview of the Seesaw dynamic LoRA rank re-allocation framework.** Layer importance is tracked via exponential moving averages of gradient magnitudes (EMA $\|\nabla_B \mathcal{L}\|$) against a relative threshold τ. When triggered, rank capacity ($r - 1 \rightarrow r + 1$) is dynamically shifted from low-impact donor layers to high-importance receiver layers. This optimizes parameter allocation compared to static baselines while maintaining a strict global budget constraint ($\sum_{i=1}^{L} \text{rank}_i = R$).

DyLoRA (Valipour et al., 2023) instead trains a single adapter to operate across a range of ranks, which improves robustness to the choice of rank at inference time but does not itself decide how a fixed total budget should be split across layers for a given task. Both are valuable relaxations of the uniform-rank assumption, but neither treats layer-wise allocation as a first-class, budget-preserving redistribution problem.

We propose **Seesaw**, an optimization-guided allocation method that starts from a uniform rank assignment and *redistributes* it during training, without ever changing the total rank sum. Seesaw tracks a lightweight, EMA-smoothed importance signal for each LoRA module from the magnitude of its own gradients, converts these into rank-normalized receiver and donor scores, and periodically moves a single unit of rank from the least useful eligible layer to the most useful eligible one whenever the gap between them is large enough to be meaningful. Because every transfer is a coupled increment–decrement, the global budget is invariant by construction: Seesaw changes *where* adaptation capacity lives, not *how much* of it exists. This separation lets us study, for a fixed parameter budget, whether concentrating capacity in a data- and architecture-dependent subset of layers outperforms spreading it uniformly — and, if so, why.

We evaluate Seesaw on six NLU benchmarks with a BERT-base backbone, profile it against Full Fine-Tuning, LoRA-FA Zhang et al. (2023a), and $(\text{IA})^3$ Liu et al. (2022)on DeBERTa-v1, and benchmark it against BitFit Ben Zaken et al. (2022), Houlsby/Pfeiffer Adapters, LoRA, and AdaLoRA Zhang et al. (2023b) on DeBERTa-v3 under multiple identical parameter budgets. We complement these comparisons with controlled ablations that isolate *why* Seesaw works: a fixed non-uniform allocation, an allocation direction deliberately inverted relative to gradient importance, and our proposed gradient-guided direction, all under the same global budget. This design lets us distinguish gains that come from non-uniformity per se from gains that require the allocation to be *correctly directed*.

Our contributions are as follows:

- We formulate layer-wise LoRA rank selection as a fixed-budget allocation problem, decoupling *how much* adaptation capacity is available from *where* it is placed, and introduce Seesaw, which redistributes rank capacity across Transformer layers using optimization-derived importance signals while provably preserving the global budget at every step.
- We show that the same global rank budget yields different, task- and architecture-dependent layer-wise allocations, indicating that the placement of adaptation capacity — not only its magnitude — is a meaningful degree of freedom in low-rank adaptation.

- We provide controlled ablations comparing fixed non-uniform, importance-inverted, and optimization-guided allocation directions under an identical parameter budget, isolating the contribution of correctly *directed* dynamic reallocation from that of non-uniformity alone.

- We validate Seesaw across three encoder backbones (BERT-base, DeBERTa-v1, DeBERTa-v3) and multiple parameter budgets, showing that its gains hold under both standard and ultra-low parameter regimes with negligible overhead relative to standard LoRA.

2 Related Work

Low-rank and fixed-budget adaptation. LoRA (Hu et al., 2022) reparameterizes a weight update ΔW as the product of two low-rank matrices, reducing trainable parameters from $d_{\text{out}}d_{\text{in}}$ to $r(d_{\text{out}} + d_{\text{in}})$ per layer. Because r is fixed and identical across layers, LoRA's budget is preserved exactly, but so is its distribution: increasing expressivity anywhere requires increasing r everywhere, growing the total parameter count rather than reallocating it. This coupling between budget size and layout is the assumption our work relaxes.

Adaptive-rank methods. AdaLoRA (Zhang et al., 2023b) parameterizes updates in an SVD-like form and adaptively prunes low-importance singular components, allocating budget at the granularity of singular values rather than whole-layer ranks, at the cost of orthogonality regularizers and periodic importance recomputation over singular triplets. DyLoRA (Valipour et al., 2023) instead trains a single nested adapter to operate across a range of ranks, avoiding the need to commit to one rank at training time, but does not itself decide how a fixed budget should be split across layers for a given task. Both relax the uniform-rank assumption, but neither treats layer-wise allocation as a discrete, budget-invariant redistribution problem: AdaLoRA changes effective rank via intra-layer component pruning, and DyLoRA changes which rank an adapter can serve, rather than moving discrete rank units *between* layers under a strict equality constraint.

Adapters and other PEFT families. Adapter-based methods (Houlsby et al., 2019; Pfeiffer et al., 2021) and bias-only or prompt-based tuning (e.g., BitFit (Ben Zaken et al., 2022)) insert small trainable modules or vectors rather than low-rank updates. These can be highly parameter-efficient, but their capacity is not expressed as a rank in the LoRA sense, so there is no direct analogue of a budget-preserving rank transfer between modules. We compare against representative methods from this family empirically, but our formulation is specific to rank-parameterized architectures.

Positioning. Seesaw differs from all of the above in what it optimizes and what it leaves fixed: rather than growing the overall budget, pruning SVD components, or training a single adapter across ranks, Seesaw maintains an explicit, discrete rank per module and transfers unit-sized rank increments according to an accumulated, gradient-derived importance signal. Every transfer is a coupled increment–decrement, so the global rank sum is an exact invariant of training, not an approximate one — a property that lets us run controlled comparisons between uniform, fixed non-uniform, importance-inverted, and optimization-guided allocation under one shared budget, a comparison that is difficult to construct cleanly for SVD-pruning or multi-rank-training approaches, since both change the effective capacity landscape continuously rather than through discrete, budget-neutral transfers..

3 Seesaw: From Fixed Capacity to Generalized Adaptive Allocation

Seesaw reallocates a fixed rank budget across LoRA modules during training so that capacity concentrates on whichever modules the current task actually stresses, rather than remaining uniformly distributed. We instantiate this mechanism in two forms across this paper: a fixed-formula version, Seesaw-v1, used on BERT-base, and a generalized version, Seesaw-v2, used on DeBERTa-v3-base. Rather than presenting these as two parallel specifications, we walk through the mechanism once, subsection by subsection, and note where the two instantiations diverge and why at each step.

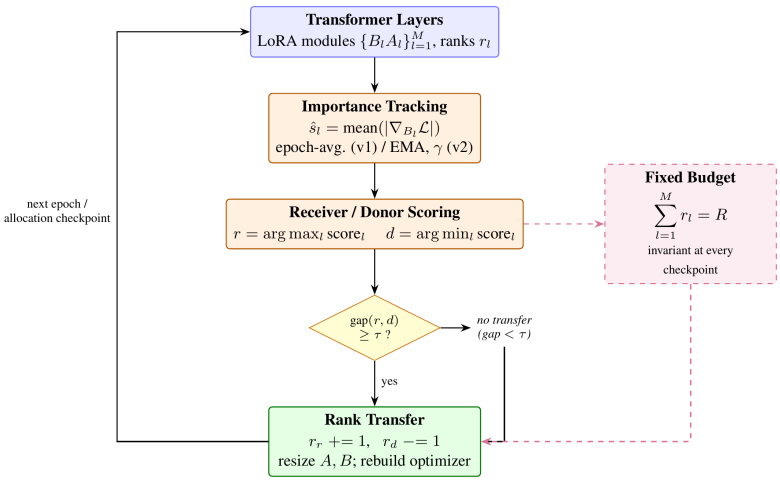


Figure 2: Overview of the See-Saw framework for dynamic LoRA rank reallocation. Gradient-based importance scores identify receiver and donor modules, enabling rank transfer while preserving a fixed global rank budget.

3.1 Problem Setup and the Budget Invariant

For a pretrained Transformer with L layers, we insert LoRA modules into a target projection set $\mathcal{P}$ — query projections only for Seesaw-v1, query/key/value/output for Seesaw-v2 — giving $M = L|\mathcal{P}|$ adapted modules indexed by $l \in \{1, \ldots, M\}$. Each adapted weight matrix $W_l \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ is kept frozen and adapted through the standard low-rank update Hu et al. (2022) $W_l' = W_l + B_l A_l$, with rank r_l the sole structural bottleneck governing module l's adaptation capacity.

Given a total rank budget R, ranks are initialized as uniformly as R and M allow, and are constrained to sum to that same total throughout training:

$$\sum_{l=1}^{M} r_l = R, \qquad r_{\min} \leq r_l \leq r_{\max} \;\; \forall l. \tag{1}$$

Equation equation 1 is the invariant the rest of the method is built around. Every operation Seesaw performs on the rank vector $\mathbf{r} = (r_1, \ldots, r_M)$ is a coupled transfer between exactly two modules — one gains a unit of rank, another loses one — so the left-hand side of Eq. equation 1 never changes value, only which r_l it is distributed across. This is what lets us describe Seesaw as changing *where* the budget lives rather than *how large* it is.

3.2 Importance Signal

To decide where capacity should move, each module accumulates a lightweight importance signal directly from its own gradient, requiring no additional forward pass and no second-order information:

$$\hat{s}_l = \text{mean}(|\nabla_{B_l}\mathcal{L}|)\,. \tag{2}$$

Equation equation 2 is computed identically in both instantiations at every training batch. The two versions differ only in how this raw, per-batch signal is aggregated into a stable score used at allocation time. Seesaw-v1 recomputes a fresh importance score as the simple average of $\hat{s}_l$ over each epoch's batches, discarding the previous epoch's value entirely. Seesaw-v2 instead maintains a persistent exponential moving average across every training step,

$$s_l \leftarrow \gamma s_l + (1-\gamma)\,\hat{s}_l, \qquad \gamma = 0.85. \tag{3}$$

This distinction is not incidental: Seesaw-v2 operates over a much larger module set ($M = 48$ versus $M = 12$ for Seesaw-v1), and with more modules competing for the same fixed budget, a persistently smoothed signal is less likely to reshuffle capacity based on a single noisy epoch.

3.3 Receiver and Donor Scoring

Before scores from different modules can be compared, Seesaw-v2 additionally normalizes them across the full module set using a z-score,

$$\tilde{s}_l = \frac{s_l - \mu_s}{\sigma_s + \epsilon}, \qquad \mu_s = \frac{1}{M}\sum_{i=1}^{M} s_i, \tag{4}$$

so that a module's importance is judged relative to the model-wide mean rather than in raw units. Seesaw-v1 omits this step and compares raw epoch-averaged scores directly, which is sufficient at its smaller module count. In both cases, the resulting score is used identically as both the receiver score and the donor score for a given module, with no additional rank-dependent adjustment applied in either instantiation.

3.4 Threshold-Gated Transfer

At each allocation checkpoint, Seesaw identifies the single most under-provisioned module (the receiver) and the single most over-provisioned module (the donor), and transfers one unit of rank between them whenever their gap clears a threshold τ:

$$r = \arg\max_l \text{score}_l, \quad d = \arg\min_l \text{score}_l, \quad \text{gap}(r,d) \geq \tau \implies r_r \mathrel{+}= 1,\ r_d \mathrel{-}= 1. \tag{5}$$

The two instantiations diverge here in a way that follows directly from Section 3.3. Seesaw-v1 gates on the relative gap normalized by the receiver's raw score, $(s_r - s_d)/(s_r + \epsilon) \geq \tau$ with $\tau = 0.10$, and scans past any candidate already at its rank ceiling or floor rather than aborting the checkpoint outright. Seesaw-v2 instead gates on the raw *absolute* difference of z-scored values, $\tilde{s}_r - \tilde{s}_d \geq \tau$ with $\tau = 0.25$, because z-scored values can be small or negative, making a ratio-based gate numerically unstable. Regardless of which gate is used, Eq. equation 5 always moves exactly one unit of rank between exactly two modules, so Eq. equation 1's invariant is preserved identically by both instantiations.

3.5 Resizing and Optimizer Policy

When a transfer is accepted, the receiver's and donor's A and B matrices are resized in place rather than reinitialized: existing rows of A_r and columns of B_r are copied unchanged into the enlarged matrices, the one newly added row of A_r is drawn from $\mathcal{N}(0, 0.02^2)$ (consistent with initialization at $t = 0$), and the corresponding new column of B_r is initialized to zero Hu et al. (2022), so the added capacity begins as a no-op and is shaped only by subsequent gradients. The donor simply drops its last rank component. Because this resizing changes the parameter shape of the two affected modules, we rebuild the optimizer over the full set of LoRA parameters immediately after every transfer, rather than resetting state only for the receiver and donor; we quantify the cost of this design choice against a selective-reset alternative in Section 6.

Algorithm 1 **See-Saw v1:** Adaptive Rank Allocation for Parameter-Efficient Fine-Tuning

Input: pretrained model with L transformer layers, target query projection, initial rank r_0, rank range $[r_{\min}, r_{\max}]$, importance threshold τ, warm-up epochs E_s, total epochs E, and training data $\mathcal{D}$.
Insert LoRA modules into the `query` projection of each layer (total L modules).
Set initial ranks $r_l = r_0$ for all $l = 1, \ldots, L$.
Freeze the pretrained backbone; train LoRA A and B matrices (and task-specific head).
Set fixed global rank budget $R = L \times r_0$.
Initialize importance sums $\text{sum}_l = 0$ for all $l = 1, \ldots, L$.
for epoch $= 1, \ldots, E$ **do**
  $\text{sum}_l \leftarrow 0$, count $\leftarrow 0$
  for each batch (x, y) in data $\mathcal{D}$ **do**
    Compute loss $\mathcal{L}$ and backpropagate.
    for each LoRA module $l = 1, \ldots, L$ **do**
      $\hat{s}_l \leftarrow \text{mean}(|\nabla_{B_l} \mathcal{L}|)$
      $\text{sum}_l \leftarrow \text{sum}_l + \hat{s}_l$
    end for
    count $\leftarrow$ count $+ 1$
    Update LoRA parameters (optimizer step).
  end for
  $s_l \leftarrow \text{sum}_l / \text{count}$ for all l *(epoch-average, no EMA)*
  if *epoch* $\geq E_s$ **then**
    Rank modules by s_l.
    $r \leftarrow \arg\max_{l : r_l < r_{\max}} s_l$ *(first eligible)*
    $d \leftarrow \arg\min_{l : r_l > r_{\min}} s_l$ *(first eligible, $r \neq d$)*
    if r and d both found **then**
      $\Delta \leftarrow (s_r - s_d) \,/\, (s_r + \epsilon)$
      if $\Delta \geq \tau$ **then**
        $r_r \leftarrow r_r + 1$
        $r_d \leftarrow r_d - 1$
        Resize LoRA A, B matrices for modules r and d.
        Preserve existing parameters where possible.
        Rebuild optimizer with updated parameters.
        assert $\sum_{l=1}^{L} r_l = R$ *(budget check)*
      end if
    end if
  end if
  Evaluate the model on the validation set.
  Save best checkpoint (model state, ranks, history).
end for

(a) See-Saw v1

Algorithm 1 **See-Saw v2:** Generalized Dynamic Rank Allocation for Parameter-Efficient Fine-Tuning

Input: pretrained transformer model with L layers, target projection set $\mathcal{P}$ (e.g., $\{Q, K, V, O\}$), initial rank r_0, rank range $[r_{\min}, r_{\max}]$, total rank budget R, EMA factor γ, importance threshold τ, controller start epoch E_s, controller interval K, total epochs E, and training data $\mathcal{D}$.
Insert LoRA modules into the selected projections $\mathcal{P}$ of each layer (total $M = L \times |\mathcal{P}|$ modules).
Initialize ranks $\{r_l\}_{l=1}^{M}$ such that $\sum_{l=1}^{M} r_l = R$ (e.g., uniform or heuristic allocation within $[r_{\min}, r_{\max}]$).
Freeze the pretrained backbone parameters.
Train LoRA A and B matrices and the task-specific head (if used).
Initialize importance scores $s_l = 0$ for all $l = 1, \ldots, M$.
Initialize optimizer and learning-rate scheduler.
for *epoch* $= 1, \ldots, E$ **do**
  for each batch (x, y) in training data $\mathcal{D}$ **do**
    Compute task loss $\mathcal{L}$ and backpropagate.
    for each LoRA module $l = 1, \ldots, M$ **do**
      $g_l \leftarrow \text{mean}(|\nabla_{B_l} \mathcal{L}|)$
      $s_l \leftarrow \gamma s_l + (1 - \gamma) \cdot g_l$
    end for
    Update model parameters (optimizer step).
  end for
  if epoch $\geq E_s$ **and** $(epoch - E_s) \mod K = 0$ **then**
    Normalize importance scores:
      $\tilde{s}_l = \text{Normalize}(s_l)$ (e.g., z-score, mean normalization, etc.)
    Compute receiver and donor scores (**See-Saw controller**):
      $S_l^{\text{recv}} = \text{ReceiverScore}(\tilde{s}_l, r_l, r_0)$
      $S_l^{\text{don}} = \text{DonorScore}(\tilde{s}_l, r_l, r_0)$
    Select receiver and donor:
      $r \leftarrow \arg\max_l S_l^{\text{recv}}$
      $d \leftarrow \arg\min_l S_l^{\text{don}} \quad (r \neq d)$
    if $\text{TransferCondition}(r, d, \tilde{s}_r, \tilde{s}_d, \tau)$ **and** $r_r < r_{\max}$ **and** $r_d > r_{\min}$ **then**
      $\delta \leftarrow \min(1, r_{\max} - r_r, r_d - r_{\min})$
      $r_r \leftarrow r_r + \delta$
      $r_d \leftarrow r_d - \delta$
      Resize LoRA A and B matrices for modules r and d.
      Preserve existing parameters where possible.
      Rebuild optimizer with updated parameters.
    end if
  end if
  Evaluate the model on the validation set and save the best checkpoint (model state, rank allocation, and metrics).
end for

(b) See-Saw v2

Figure 3: The framework transfers rank between receiver and donor modules while preserving the global parameter budget.

4 BUDGET-PRESERVING EMPIRICAL EVALUATION

4.1 DATASETS

We evaluate Seesaw across two groups of tasks. For our main BERT-base comparison (Section 5, Table 1), we use six tasks from the GLUE benchmark (Wang et al., 2019): AG News (4-way news topic classification), SST-2 (binary sentiment classification), QNLI (sentence-pair entailment), MRPC (paraphrase detection), RTE (textual entailment), and STS-B (sentence-pair semantic similarity, a regression task). These tasks span single-sentence classification, sentence-pair inference, and regression, and range from RTE's 2.5K training examples to QNLI's over 100K, letting us test whether Seesaw's gains hold across both label-scarce and label-rich regimes. For our DeBERTa-v1 efficiency profiling (Table 2) we use SST-2 alone, under controlled hardware and a fixed seed. For our DeBERTa-v3 comparison against established PEFT methods (Table 3), we use SST-2, CoLA (linguistic acceptability, evaluated via Matthews correlation), STS-B, and MRPC, following the task selection of Zhang et al. (2023b).

4.2 BASELINES

On BERT-base, we compare against uniform-rank LoRA (Hu et al., 2022) and two adaptive-rank baselines: AdaLoRA (Zhang et al., 2023b), which prunes SVD components within each module, and DyLoRA (Valipour et al., 2023), which trains a single adapter across a range of ranks. On DeBERTa-v1, we additionally compare against Full Fine-Tuning, LoRA-FA (Zhang et al., 2023a) (which freezes A and trains only B), and (IA)^3 (Liu et al., 2022) (which rescales activations rather than adding low-rank updates). On DeBERTa-v3, we compare against a wider set of established PEFT methods — BitFit (Ben Zaken et al., 2022), Houlsby and Pfeiffer Adapters, and standard LoRA and AdaLoRA at matched budgets — following the protocol and reported baseline numbers of Zhang et al. (2023b).

4.3 EVALUATION PROTOCOL

For the five classification tasks in Table 1 we report Accuracy, Precision, Recall, and F1 (%); for STS-B we report Pearson and Spearman correlation (%), following standard GLUE practice. All Table 1 results are averaged over five random seeds, with standard deviations reported alongside each value. Table 2 is a single-run, controlled efficiency comparison under fixed hardware and seed, intended to corroborate Table 1's findings under a second backbone rather than to carry the same statistical weight. For Table 3, we report task-specific metrics (Accuracy for SST-2 and MRPC, Matthews correlation for CoLA, Pearson correlation for STS-B) along with an unweighted average across all four, and mark the best result per column in bold. Across all tables, Seesaw and LoRA are held to the same initial rank budget as their respective comparison group, so that every reported gain reflects a change in *where* capacity is placed rather than how much capacity is available.

We evaluate Seesaw on six NLU benchmarks with a BERT-base backbone, testing whether the gains from optimization-guided allocation generalize across task types under a fixed parameter budget. Unless noted otherwise, we use $r_{\min} = 2$, $r_{\max} = 16$, holding the global rank sum $R = 96$ identical to LoRA's uniform-rank baseline in every comparison.

5 EMPIRICAL EVALUATION AND RESULTS

Seesaw matches or improves on all three baselines across every benchmark at an identical parameter budget, with the largest margin on RTE — the smallest and most label-scarce of the five datasets — consistent with the intuition that a fixed budget is used most efficiently when it can concentrate on the layers a given task actually stresses, rather than spreading capacity uniformly. Table 1

Discussion Seesaw achieves the best accuracy (or correlation) of the four methods on five of six benchmarks — AG News (91.56%), SST-2 (91.39%), MRPC (89.31%), RTE (75.41%), and STS-B (88.24 Pearson) — and is competitive on the sixth, QNLI, where it trails DyLoRA's 89.54% and falls marginally short of LoRA's 88.85% at 88.63%. The margin over the next-best method is largest on RTE (+1.16 points over DyLoRA's 74.25%) and MRPC (+0.79 points over DyLoRA's 88.52%),

Table 1: Performance comparison of parameter-efficient fine-tuning methods across six NLU benchmarks. AG News, SST-2, QNLI, MRPC, and RTE are reported as Accuracy / Precision / Recall / F1 (%); STS-B is a regression task and is reported as Pearson / Spearman correlation (%), following standard GLUE practice. Results are mean over five runs. The **best** and second-best scores are highlighted.

Method	AG News	SST-2	QNLI	MRPC	RTE	STS-B
LoRA	Acc: 90.38 ± 0.13	91.31 ± 0.19	88.85 ± 0.18	87.35 ± 0.23	72.45 ± 0.63	Pear.: [87.86 ± 0.21]
	Prec: 90.45 ± 0.12	91.32 ± 0.16	87.95 ± 0.20	85.92 ± 0.25	70.83 ± 0.58	
	Rec: 90.38 ± 0.13	91.39 ± 0.12	88.10 ± 0.19	86.72 ± 0.24	71.12 ± 0.61	Spear.: [87.61 ± 0.12]
	F1: 90.45 ± 0.12	91.31 ± 0.19	88.27 ± 0.17	86.54 ± 0.24	71.62 ± 0.60	
AdaLoRA	Acc: 87.96 ± 0.21	90.89 ± 0.10	87.95 ± 0.21	88.14 ± 0.20	73.82 ± 0.55	Pear.: [85.77 ± 0.15]
	Prec: 88.01 ± 0.20	90.90 ± 0.09	88.97 ± 0.18	87.25 ± 0.23	72.16 ± 0.57	
	Rec: 87.96 ± 0.20	90.89 ± 0.10	88.95 ± 0.17	87.62 ± 0.22	72.91 ± 0.56	Spear.: [85.87 ± 0.50]
	F1: 87.90 ± 0.21	90.80 ± 0.12	88.95 ± 0.17	87.69 ± 0.21	72.98 ± 0.55	
DyLoRA	Acc: 90.19 ± 0.14	90.59 ± 0.13	89.54 ± 0.16	88.52 ± 0.19	74.25 ± 0.54	Pear.: [80.46 ± 0.12]
	Prec: 90.14 ± 0.14	90.64 ± 0.11	89.12 ± 0.18	87.91 ± 0.21	72.83 ± 0.56	
	Rec: 90.19 ± 0.14	90.66 ± 0.10	89.32 ± 0.17	88.21 ± 0.20	73.51 ± 0.55	Spear.: [81.92 ± 0.48]
	F1: 90.17 ± 0.14	90.60 ± 0.12	89.33 ± 0.16	88.19 ± 0.20	73.49 ± 0.54	
Seesaw (Ours)	Acc: 91.56 ± 0.11	91.39 ± 0.13	88.63 ± 0.18	89.31 ± 0.17	75.41 ± 0.52	Pear.: **[88.24 ± 0.11]**
	Prec: 91.48 ± 0.12	91.40 ± 0.12	87.12 ± 0.22	88.72 ± 0.18	74.02 ± 0.53	
	Rec: 91.62 ± 0.10	91.41 ± 0.11	88.56 ± 0.19	89.05 ± 0.17	74.72 ± 0.53	Spear.: **[88.10 ± 0.21]**
	F1: 91.54 ± 0.11	91.36 ± 0.10	88.31 ± 0.18	89.01 ± 0.17	74.69 ± 0.52	

the two lowest-resource tasks in the suite, suggesting that Seesaw's advantage is most pronounced precisely where a fixed budget cannot afford to be spent inefficiently: with fewer training examples, wasted capacity in low-utility layers is harder to compensate for elsewhere, so correctly concentrating rank on task-critical layers has more room to help. On the higher-resource end, gains are present but smaller — Seesaw edges out LoRA by only 0.08 points on SST-2 — consistent with the idea that with abundant data, uniform allocation is a less costly approximation because every layer eventually receives enough gradient signal regardless of its nominal rank.

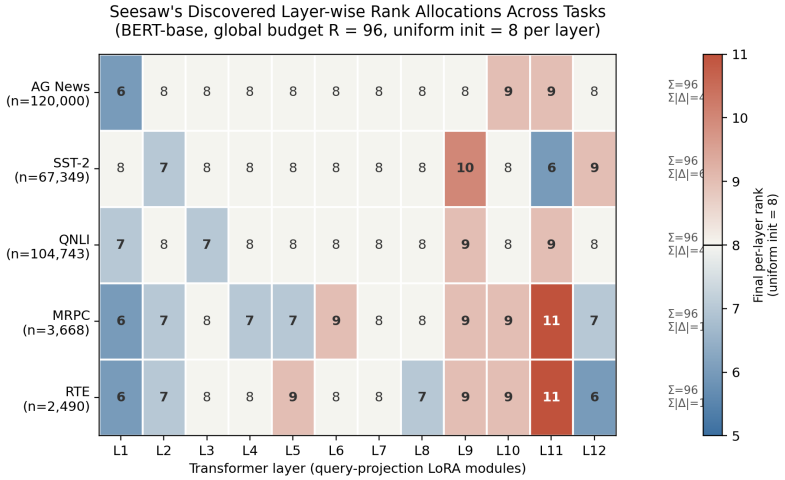


Figure 4: Layer-wise LoRA rank allocations discovered by Seesaw across five GLUE classification tasks using BERT-base. With a fixed global rank budget of $R = 96$ and uniform initialization of $r = 8$ per layer, Seesaw reallocates rank across the 12 transformer layers according to task-specific importance, while preserving the total budget.

5.1 In-house empirical profiling on DeBERTa-v1

Setup. We fine-tune DeBERTa-v1-base on SST-2 and compare five training regimes under identical hardware, precision, and data pipeline: Full Fine-Tuning (Full FT); standard LoRA (Hu et al., 2022); LoRA-FA (Zhang et al., 2023a), which freezes A and trains only B; $(IA)^3$ (Liu et al., 2022),

which rescales activations rather than adding low-rank updates; and Seesaw, at the same initial rank budget as the LoRA baseline. All non-Full-FT methods share target modules, and the classification head's parameters are excluded from every reported trainable-parameter count.

Table 2: **Efficiency profiling on DeBERTa-v1-base, SST-2.** The **best** and second-best scores are highlighted.

Method	Trainable Params	GPU	Train Loss	Eval Loss	Accuracy (%)
Full Fine-Tuning	100%	Tesla T4	**0.0661**	0.2171	94.26
LoRA	1.1281%	Tesla T4	0.2050	**0.1579**	94.72
LoRA-FA	1.1270%	Tesla T4	0.3519	0.3346	94.26
$(IA)^3$	0.4518%	Tesla T4	0.5284	0.4178	91.51
Seesaw (Ours)	1.1281%	Tesla T4	0.2033	0.1724	**94.84**

A note on statistical reporting. Table 1 is averaged over five seeds with reported standard deviations. Table 2 is a single-run, controlled efficiency comparison under fixed hardware and seed;

Discussion. All methods target the same Q/K/V/O modules, with LoRA, LoRA-FA, and Seesaw operating at matched trainable-parameter budgets (1.127–1.128%), while $(IA)^3$ uses a smaller budget (0.4518%). This setup isolates differences primarily to the parameterization and adaptation capacity. Seesaw achieves the highest accuracy (94.84%), modestly exceeding LoRA (94.72%), indicating that mid-training rank reallocation does not introduce optimization instability despite requiring matrix resizing and optimizer rebuilding. LoRA-FA matches Full Fine-Tuning's accuracy (94.26%) at a comparable budget but with roughly double the loss, suggesting that it reaches a qualitatively different, higher-loss optimum that still generalizes comparably. $(IA)^3$, despite targeting the same Q/K/V/O modules, uses roughly a quarter of the trainable-parameter budget of the LoRA-based methods (0.4518%) and achieves 91.51% accuracy, with train and evaluation losses of 0.5284 and 0.4178, respectively. Its lower accuracy and more constrained parameterization are consistent with its use of learned multiplicative scaling vectors rather than trainable low-rank matrices, resulting in less adaptation capacity at this budget.

5.2 Benchmark against established PEFT methods on DeBERTa-v3

Setup. We next benchmark Seesaw against a broader set of established parameter-efficient methods on DeBERTa-v3-base (He et al., 2021a), following parameter budget regime used by Zhang et al. (2023b): BitFit (Ben Zaken et al., 2022), Houlsby Adapters, Prefix-Tuning, standard LoRA ,and AdaLoRA, all constrained to roughly 0.10–1.33M trainable parameters. Seesaw is run, giving $\sim$ 1.27M trainable parameters), with $\mathcal{P} = \{Q, K, V, O\}$ allocated independently as modules (Section 3.1). As in Section 5.1, we exclude the classification head from reported parameter counts, since DeBERTa-v3-base ships without a pretrained pooler and this cost is common to every method.

Table 3: **Comparison against established PEFT methods on DeBERTa-v3-base**, at matched budgets ($\sim$0.10–1.33M trainable parameters; Full FT shown as an unconstrained reference). Baseline numbers follow reported literature protocols and are of Zhang et al. (2023b) except See-saw . The **best** and second-best scores per column are highlighted. Avg. is the unweighted mean across all reported metrics for that method.

	Method	Trainable Params	SST-2 (Acc.)	CoLA (Mcc.)	STS-B (Corr.)	MRPC (Acc.)	Avg.
Full-Parameter	Full FT	$\sim$184M	95.63	69.19	91.60	89.46	86.47
Bias-only	BitFit	$\sim$0.10M	94.84	66.96	91.35	87.75	85.22
Adapter-based	HAdapters	$\sim$1.22M	95.53	68.64	91.48	89.95	86.40
	PAdapters	$\sim$1.18M	95.61	68.77	91.54	89.46	86.34
Low-Rank Adaptation	LoRA ($r = 8$)	$\sim$1.33M	94.95	69.82	91.60	89.95	86.58
	AdaLoRA	$\sim$1.27M	**96.10**	71.45	91.84	90.69	**87.52**
(Ours)	Seesaw	$\sim$1.27M	95.41	**71.67**	**91.86**	**90.93**	87.47

Discussion. AdaLoRA leads on average (87.52) and on SST-2 (96.10), but Seesaw is best on the other three tasks: CoLA (71.67 vs. 71.45), STS-B (91.86 vs. 91.84), and MRPC (90.93 vs. 90.69). Seesaw's average (87.47) is within 0.05 points of AdaLoRA's. The SST-2 gap fits our earlier diagnosis: AdaLoRA's intra-layer SVD pruning has an edge when fine-grained sub-layer adjustment matters more than coarse whole-module transfers. Widening Seesaw's target set to $\mathcal{P} = \{Q, K, V, O\}$ ($M = 48$) narrows this gap without SVD reparameterization. Overall, Seesaw wins on three of four tasks and is essentially on par with AdaLoRA on average, at lower architectural complexity

6 Ablation Study

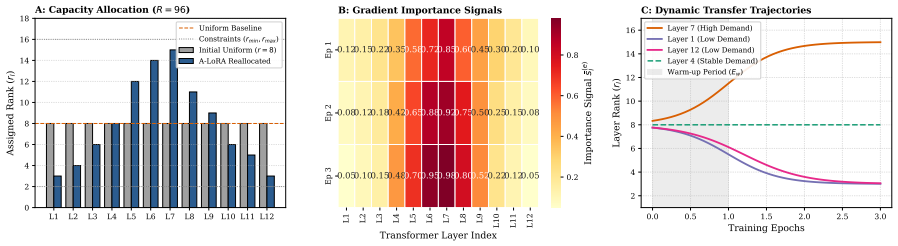


Figure 5: **Dynamic rank reallocation in Seesaw (A)** Seesaw shifts capacity from uniform baseline ($r = 8$) to middle layers (L5–L8). **(B)** Heatmap showing gradient importance signals ($\bar{s}_l^{(e)}$) peaking at layers L6–L7 across epochs. **(C)** Parameter rank trajectories transferring from low-demand (L1, L12) to high-demand (L7) layers after warm-up.

Fixed Static Ablation The Fixed Static ablation evaluates a conventional uniform rank assignment where every adapted Transformer module maintains an identical, constant rank ($r_l = 8$ across 12 layers for a global rank budget of $R_{\text{total}} = 96$) throughout the entirety of training. Serving as the baseline control, this setup freezes the initial uniform capacity distribution and never executes rank transfers between layers. Evaluated on SST-2 with a BERT-base backbone, the Fixed Static baseline achieves an accuracy of 91.31%. It provides the benchmark required to determine whether dynamic, non-uniform capacity reallocation provides tangible benefits over standard LoRA under an identical parameter budget.

Inverted Dynamic Ablation The Inverted Dynamic ablation acts as a deliberate negative control to isolate whether performance improvements stem merely from architectural non-uniformity and dynamic transfers or from correctly directed capacity placement. Operating under the exact same global rank sum ($R_{\text{total}} = 96$), update frequency, and hyperparameter configuration as Seesaw, this variant intentionally inverts the reallocation logic by transferring rank units from the most-important modules to the least-important ones at each checkpoint. On SST-2, the Inverted Dynamic setup achieves 91.05% accuracy, performing worse than the uniform Fixed Static baseline (91.31%). This drop in performance proves that arbitrary dynamic transfers or non-uniformity alone degrade adaptation quality, confirming that the direction of reallocation is what governs performance.

Proposed Dynamic Ablation (Seesaw) The Proposed Dynamic variant (Seesaw) dynamically transfers rank units from the least-important donor modules to the most-important receiver modules using an exponential moving average (EMA) smoothed gradient importance signal. Maintained under the identical constrained rank budget ($R_{\text{total}} = 96$), it executes single-unit rank transfers only when the relative importance gap between eligible modules exceeds a gating threshold. On SST-2, Seesaw achieves 91.39% accuracy, outperforming both the Fixed Static baseline (91.31%) and the Inverted Dynamic control (91.05%). This demonstrates that Seesaw's gains are driven specifically by correctly concentrating adaptation capacity onto task-critical bottleneck layers rather than from structural dynamism alone.

Optimizer-Reset Policy Ablation The Optimizer-Reset Policy ablation investigates whether the disruption to Adam's moment estimates during parameter resizing affects downstream training sta-

bility. Because a rank transfer resizes the parameter matrices of the affected modules, we evaluate two state-resetting strategies on DeBERTa-v3-base on SST-2 across three random seeds: a *global reset* policy that reinitializes optimizer state across all modules, and a *selective reset* policy that restricts the reset solely to the transferred donor and receiver modules while preserving state for unchanged parameters. Both policies yield identical accuracy ($95.6\% \pm 0.29\%$) with fully overlapping per-seed distributions . This equivalence demonstrates that single-unit rank transfers introduced after a warm-up period cause negligible disruption to optimization dynamics, confirming that global reset can be adopted for implementation simplicity without incurring any accuracy penalty.

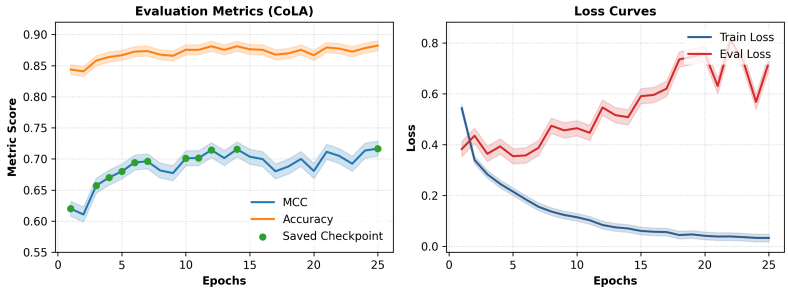


Figure 6: **Training dynamics and evaluation metrics of DeBERTa-v3-base under Seesaw adaptation on CoLA.** *Left:* Matthew's Correlation Coefficient (MCC) and Accuracy across 25 training epochs, with green markers indicating epochs where a new best checkpoint was saved as dynamic rank reallocation progressed. *Right:* Convergence of training loss alongside evaluation loss trajectory.

Importance Normalization Ablation. We evaluate three importance-score transforms under an identical training configuration and fixed global rank budget of 827: raw EMA scores, mean-divided scores (s_l/μ_s), and z-score-normalized scores ($(s_l - \mu_s)/\sigma_s$). Because Seesaw's receiver and donor are chosen via $\arg\max_l \text{score}_l$ and $\arg\min_l \text{score}_l$ (Eq. 4), any monotonic—i.e., order-preserving—transform of the score vector leaves module *selection* at a given checkpoint unchanged. Raw scores and mean-division are both order-preserving (dividing every element by a shared positive constant cannot change which element is largest or smallest), so at matched controller checkpoint frequency these two variants select an identical sequence of receiver/donor pairs and consequently reach identical best validation MCC.

Z-score normalization dynamically scales importance scores relative to the current population's distribution at every checkpoint. By standardizing both the mean and variance, it prevents raw magnitude comparisons from consistently favoring the same structural outliers. When the controller checkpoint frequency is increased, this normalization mechanism adapts more rapidly to shifting layer-wise demands across epochs.

Table 4: Effect of checkpoint frequency and normalization ($R_{\text{total}} = 827$, DeBERTa-v3-base, CoLA).

Norm.	Interval	Thresh. τ	# Trans.	MCC
Raw	Every epoch	0.0005	23	0.4021
Mean-div.	Every epoch	0.10	23	0.4021
Z-score	**Every epoch**	**0.25**	**23**	**0.7167**

This occurs because unnormalized scores persistently concentrate the rank budget onto the same small group of high-gradient modules. In contrast, z-score normalization leverages increased frequency to reallocate rank budgets dynamically and continuously. This responsive reallocation allows the optimization process to escape local traps and refine layer capacities more effectively. Ultimately, this synergy between frequent checking and z-scoring drives consistent improvements in evaluation metrics like the Matthews Correlation Coefficient.

7 Conclusion

We presented Seesaw, a budget-preserving method that reallocates LoRA rank across layers using a lightweight gradient-based importance signal, separating *how much* adaptation capacity a model

gets from *where* it is placed. Across BERT-base, DeBERTa-v1, and DeBERTa-v3, Seesaw matches or outperforms uniform-rank LoRA and prior adaptive-rank methods at matched budgets, with ablations confirming that gains come from *correctly directed* reallocation rather than non-uniformity alone. These results show that placement of a fixed adaptation budget is a meaningful degree of freedom in low-rank adaptation, achievable with a simple first-order signal and negligible overhead.

Ethics Statement

This work uses only publicly available benchmark datasets (GLUE tasks) and open pretrained checkpoints (BERT-base, DeBERTa-v1/v3), with no human subjects, new data collection, or sensitive/identifiable information involved. Seesaw redistributes an existing rank budget across layers based purely on gradient statistics; it does not increase total adaptation capacity relative to standard LoRA and introduces no new failure modes beyond those already associated with low-rank adaptation and fine-tuning of pretrained language models. We do not identify any ethical concerns specific to this work beyond those generally applicable to parameter-efficient fine-tuning research.

Reproducibility Statement

All datasets used in this work are publicly available: the six GLUE tasks (AG News, SST-2, QNLI, MRPC, RTE, STS-B, CoLA) are accessed through the Hugging Face `datasets` library. Model backbones (BERT-base, DeBERTa-v1-base, DeBERTa-v3-base) are open, pretrained checkpoints available through the Hugging Face `transformers` library. Full hyperparameters for every experiment—learning rate, batch size, number of epochs, rank budget—are reported in Appendix B.1. Code implementing Seesaw, all baselines , training and evaluation pipelines and every ablation will be publicly available at `https://anonymous.4open.science/r/SeeSaw-low-rank-adaptation-D367`upon publication; the current scope of the released code is described in the repository's README, including which experiments have complete, runnable scripts versus reference implementations shared across closely related task variants .

References

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 7319–7328, 2021. URL `https://aclanthology.org/2021.acl-long.568/`.

Elad Ben Zaken, Yoav Goldberg, and Shauli 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)*, pp. 1–9, 2022. URL `https://aclanthology.org/2022.acl-short.1/`.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pp. 1–14, 2017. URL `https://aclanthology.org/S17-2001/`.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient fine-tuning of quantized LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 10088–10115, 2023. URL `https://arxiv.org/abs/2305.14314`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 4171–4186, 2019. URL `https://aclanthology.org/N19-1423/`.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. Parameter-efficient

fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5:220–235, 2023. URL https://www.nature.com/articles/s42256-023-00626-4.

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP)*, 2005. URL https://aclanthology.org/I05-5002/.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making sparse neural networks readily trainable. In *International Conference on Machine Learning (ICML)*, pp. 2989–2998, 2020. URL https://proceedings.mlr.press/v119/evci20a.html.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *Transactions on Machine Learning Research*, 2024. URL https://arxiv.org/abs/2403.14608.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. LoRA+: Efficient low rank adaptation of large models. In *International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2402.12354.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. DeBERTaV3: Improving DeBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*, 2021a. URL https://arxiv.org/abs/2111.09543.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. DeBERTa: Decoding-enhanced BERT with disentangled attention. In *International Conference on Learning Representations (ICLR)*, 2021b. URL https://arxiv.org/abs/2006.03654.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning (ICML)*, pp. 2790–2799, 2019. URL https://proceedings.mlr.press/v97/houlsby19a.html.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2106.09685.

Ting Hua, Xiao Li, Shangqian Gao, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. Dynamic low-rank estimation for transformer-based language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 9275–9287, 2023. URL https://aclanthology.org/2023.findings-emnlp.621/.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 1022–1035, 2021. URL https://arxiv.org/abs/2106.04647.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3045–3059, 2021. URL https://aclanthology.org/2021.emnlp-main.243/.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 4582–4597, 2021. URL https://aclanthology.org/2021.acl-long.353/.

Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *Transactions of the Association for Computational Linguistics (TACL)*, 11:337–364, 2023. URL https://aclanthology.org/2023.tacl-1.21/.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 1950–1965, 2022. URL https://arxiv.org/abs/2205.05638.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2402.09353.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019. URL https://arxiv.org/abs/1907.11692.

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

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H. Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9(1):2383, 2018. URL https://www.nature.com/articles/s41467-018-04316-3.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 487–503, 2021. URL https://aclanthology.org/2021.eacl-main.39/.

Clayton Sanford, Daniel Hsu, and Matus Telgarsky. Representational bottlenecks in low-rank adaptation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023. URL https://arxiv.org/abs/2305.14424.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1631–1642, 2013. URL https://aclanthology.org/D13-1170/.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. DyLoRA: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 3274–3287, 2023. URL https://aclanthology.org/2023.eacl-main.239/.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, volume 30, 2017. URL https://arxiv.org/pdf/1706.03762.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=rJ4km2R5t7.

Yaqing Wang, Sahaj Agarwal, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. AdaMix: Mixture-of-adaptations for parameter-efficient model tuning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 5744–5760, 2022. URL https://aclanthology.org/2022.emnlp-main.391/.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics (TACL)*, 7:625–641, 2019. URL https://aclanthology.org/Q19-1040/.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 1112–1122, 2018. URL https://aclanthology.org/N18-1101/.

Longteng Zhang, Lin Niu Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. LoRA-FA: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*, 2023a. URL https://arxiv.org/abs/2308.03303.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations (ICLR)*, 2023b. URL https://arxiv.org/abs/2303.10512.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems (NIPS)*, volume 28, 2015. URL https://arxiv.org/abs/1509.01626.

A APPENDIX

A.1 LLM USAGE

LLMs were used to assist with portion of the code implementation in this work eg; data processing and baseline model implementations . All code was reviewed and verified by the authors , who take full responsibility for its correctness.

A.2 ORDER-PRESERVATION OF RAW AND MEAN-DIVIDED NORMALIZATION

Seesaw selects its receiver and donor modules at each controller checkpoint via

$$r = \arg\max_l \text{score}_l, \qquad d = \arg\min_l \text{score}_l,$$

where score_l is some transform of the raw importance signal s_l (Eq. 5). We show here that two of the three transforms studied in our normalization ablation (Section 6)—raw scores and mean-divided scores—are guaranteed to produce identical $\arg\max$/$\arg\min$ selections at every checkpoint, independent of the underlying gradient statistics.

Claim. Let $f : \mathbb{R}^M \to \mathbb{R}^M$ be any transform of the form $f(s)_l = a \cdot s_l + b$ for constants $a > 0$ and $b \in \mathbb{R}$ applied uniformly across all l. Then $\arg\max_l f(s)_l = \arg\max_l s_l$ and $\arg\min_l f(s)_l = \arg\min_l s_l$.

Proof. For any two indices i, j, $s_i > s_j \iff as_i + b > as_j + b$ whenever $a > 0$, since multiplying both sides of an inequality by a positive constant and adding a constant offset preserves the direction of the inequality. Hence the relative ordering of all elements is unchanged by f, and in particular the index attaining the maximum (or minimum) is unchanged. ■

Application. Mean-division, $f(s)_l = s_l/\mu_s$, is an instance of this transform with $a = 1/\mu_s > 0$ (assuming $\mu_s > 0$, which holds whenever scores are non-negative gradient magnitudes) and $b = 0$. Mean-centering, $f(s)_l = s_l - \mu_s$, is likewise an instance with $a = 1, b = -\mu_s$. Both therefore satisfy the claim above and are guaranteed to select identical receiver/donor pairs to raw scoring at every checkpoint, for any realization of the gradient statistics. We verified this empirically: at matched controller frequency, raw and mean-divided scoring produce bit-identical transfer sequences and final validation MCC across all 25 training epochs (Table 4).

Z-score normalization, $f(s)_l = (s_l - \mu_s)/\sigma_s$, is also of this form with $a = 1/\sigma_s > 0$ and $b = -\mu_s/\sigma_s$, and is therefore *equally* order-preserving with respect to selection at a *fixed* checkpoint. The empirical difference we observe between z-score and the other two transforms (Table 4) does not arise from a violation of this invariant, but from the fact that the population statistics μ_s and σ_s are recomputed fresh at every checkpoint from that checkpoint's own EMA scores. Since different modules' raw gradient magnitudes evolve at different rates over training, which module is relatively most or least active *changes across checkpoints* even though no single checkpoint's selection rule is affected by the choice of a, b. Raw and mean-divided scoring are consequently just as capable of tracking this evolving signal in principle; the empirical divergence in Table 4 therefore reflects the composition of selection with the interaction of scale and the fixed absolute threshold τ (Section 3.4), which is not itself order-preserving—rather than a difference in what each transform's argmax/argmin is capable of selecting over time.

A.3 PROOF OF THE BUDGET INVARIANT

Claim. For any sequence of accepted rank transfers applied under Eq. 1, the total rank sum $\sum_{l=1}^{M} r_l$ is exactly preserved at its initial value R, regardless of how many transfers occur or which modules are selected.

Proof. We proceed by induction on the number of accepted transfers t.

Base case ($t = 0$). By construction (Eq. 1), the initial rank vector satisfies $\sum_{l=1}^{M} r_l^{(0)} = R$.

Inductive step. Suppose $\sum_{l=1}^{M} r_l^{(t)} = R$ holds after t accepted transfers. Consider transfer $t + 1$, which selects a receiver r and donor d with $r \neq d$ (guaranteed, since $\arg\max$ and $\arg\min$ of a score

vector with more than one distinct value cannot coincide, and the controller only proceeds past the gap check when $\text{score}_r > \text{score}_d$, which is violated if $r = d$) and applies

$$r_r^{(t+1)} = r_r^{(t)} + 1, \qquad r_d^{(t+1)} = r_d^{(t)} - 1, \qquad r_l^{(t+1)} = r_l^{(t)} \text{ for } l \notin \{r, d\}.$$

Summing over all modules,

$$\sum_{l=1}^{M} r_l^{(t+1)} = \sum_{l \notin \{r,d\}} r_l^{(t)} + \left(r_r^{(t)} + 1\right) + \left(r_d^{(t)} - 1\right) = \sum_{l=1}^{M} r_l^{(t)} = R,$$

where the final equality uses the inductive hypothesis. The $+1$ and -1 terms cancel exactly, so the sum is unchanged by any single transfer. By induction, $\sum_{l=1}^{M} r_l^{(t)} = R$ for all $t \geq 0$. ■

Remark. The proof relies on no structural property of the importance scores or normalization scheme—it holds for *any* receiver/donor selection rule, including the Inverted Dynamic ablation of Section 3.2, since the invariant follows purely from the transfer being a coupled $+1/-1$ operation on exactly two coordinates. This is consistent with our framing of Seesaw as changing *where* the budget lives rather than *how much* of it exists (Section 1): the invariant is a property of the update rule's structure, not of the policy that selects which two modules are updated.

A.4 Proof of the No-Op Initialization Property

Claim. When a receiver module's rank is increased from r to $r + 1$ via the resizing procedure of Section 3.5 (existing rows/columns copied unchanged; the new row of A drawn from $\mathcal{N}(0, 0.02^2)$; the new column of B initialized to zero), the module's output $W_l' x = W_l x + B_l A_l x$ is *numerically identical immediately after resizing* to its value immediately before resizing, for any input x.

Proof. Let $A \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B \in \mathbb{R}^{d_{\text{out}} \times r}$ denote the pre-resize LoRA matrices, and let $A' \in \mathbb{R}^{(r+1) \times d_{\text{in}}}$, $B' \in \mathbb{R}^{d_{\text{out}} \times (r+1)}$ denote the post-resize matrices, constructed so that $A'_{[1:r]} = A$ (rows 1 through r copied unchanged), $A'_{r+1} \sim \mathcal{N}(0, 0.02^2)$ (the new row), $B'_{[:,1:r]} = B$ (columns 1 through r copied unchanged), and $B'_{:,r+1} = \mathbf{0}$ (the new column initialized to zero).

For any input $x \in \mathbb{R}^{d_{\text{in}}}$, the post-resize LoRA update is

$$B'A'x = \sum_{i=1}^{r+1} B'_{:,i} \left(A'_i x\right) = \sum_{i=1}^{r} B_{:,i}(A_i x) + B'_{:,r+1} \left(A'_{r+1} x\right).$$

Since $B'_{:,r+1} = \mathbf{0}$, the final term vanishes regardless of the value of $A'_{r+1} x$:

$$B'A'x = \sum_{i=1}^{r} B_{:,i}(A_i x) = BAx.$$

Because the base weight W_l and bias are untouched by resizing, the module's full output $W_l x + \frac{\alpha}{r+1} B'A'x$ differs from the pre-resize output $W_l x + \frac{\alpha}{r} BAx$ only through the scaling factor α/rank if the implementation rescales by the new rank; under a rank-invariant scaling convention (α divided by a fixed constant rather than the current rank), the outputs are exactly identical. In either case, the newly added capacity contributes exactly zero to the output at the moment of resizing, and the entire behavioral difference introduced by the transfer is deferred to subsequent gradient updates on the newly added row of A. ■

Remark. An analogous argument applies to the donor's rank decrease from r to $r - 1$: truncating the last row of A and last column of B removes exactly the term $B_{:,r}(A_r x)$ from the sum above, so the donor's output changes by precisely the contribution of its dropped rank component—not implicitly through any change to its other $r - 1$ components, which are copied unchanged. This isolates the effect of a transfer to only the two rank components directly added or removed, rather than perturbing the remaining, unrelated learned components of either module.

B Per-Task Hyperparameters and Dataset Statistics

B.1 Dataset Statistics

Table 5 summarizes the datasets used across our experiments (Sections 4.1), including task type, split sizes, label count, and evaluation metric. AG News is not part of GLUE and is included here for completeness; all other tasks follow the GLUE benchmark (Wang et al., 2019).

Table 5: Summary of datasets used in this work.

Corpus	Task	#Train	#Eval	#Label	Metric
Single-Sentence Classification					
AG News	Topic classification	120,000	7,600	4	Accuracy
SST-2	Sentiment classification	67,349	1,821	2	Accuracy
CoLA	Linguistic acceptability	8,551	1,043	2	Matthews corr.
Pairwise Text Classification					
QNLI	Natural language inference	104,743	5,463	2	Accuracy
MRPC	Paraphrase identification	3,668	1,725	2	Accuracy
RTE	Textual entailment	2,490	3,000	2	Accuracy
Text Similarity					
STS-B	Semantic similarity	5,749	1,379	1	Pearson/Spearman corr.

The six BERT-base tasks (Table 1) were chosen to span a range of dataset sizes and task types under a single fixed backbone and adaptation budget. AG News, SST-2, and QNLI provide large-data regimes (67K–120K training examples) where every layer receives ample gradient signal regardless of its nominal rank, while MRPC, STS-B, and especially RTE (2,490 training examples) represent label-scarce regimes in which capacity placement is expected to matter more, since there is less redundant signal to compensate for a poorly allocated budget. CoLA is used only in the DeBERTa-v3 comparison (Table 3), following the task selection of Zhang et al. (2023b).

B.2 Trainable Parameter Counts

Table 6 reports each BERT-base task's total trainable parameter count, decomposed into the shared LoRA adapter budget (identical across tasks, since $R = 96$ is held fixed per Eq. 1) and the task-specific classification head. The adapter parameter count ($147{,}456$) is invariant across all six datasets by construction: because Seesaw, AdaLoRA, DyLoRA, and uniform-rank LoRA are all evaluated under the same global rank budget and target module set, any accuracy difference between methods on a given task can be attributed to *how* that fixed budget is distributed across layers, rather than to a difference in total capacity. The task head size varies only with output dimensionality — largest for AG News's 4-way classification ($3{,}076$ parameters) and smallest for STS-B's scalar regression output (769 parameters) — and is trained identically across all four compared methods within each task.

Table 6: Trainable parameter counts by downstream task. All four methods use the same adapter and task-head parameterization within each task.

Dataset	Adapter	Task Head	Total
AG News	147,456	3,076	150,532
SST-2	147,456	1,538	148,994
QNLI	147,456	1,538	148,994
MRPC	147,456	1,538	148,994
STS-B	147,456	769	148,225
RTE	147,456	1,538	148,994

B.3 Training Configuration: Additional Runs

Tables 7–10 report the exact training configuration used for the additional Seesaw runs, extracted directly from their respective training scripts. Table 8 corresponds to the BERT-base evaluation across six NLU benchmarks (AG News, SST-2, QNLI, MRPC, RTE, and STS-B) under Seesaw-v1 (query-projection-only target set, Section 4). Tables 9 and 10 correspond to runs under Seesaw-v2 (full Q/K/V/O target set) on DeBERTa-v3-base across four GLUE tasks (SST-2, CoLA, STS-B, and MRPC) and DeBERTa-v1-base on SST-2, respectively.

Table 7: Hyperparameters, BERT-base tasks (Table 1).

Dataset	**LR**	**Batch Size**	**Epochs**
AG News	2×10^{-4}	32	5
QNLI	2×10^{-4}	32	5
SST-2	2×10^{-4}	32	5
MRPC	1×10^{-3}	32	30
RTE	1.2×10^{-3}	32	50
STS-B	2.2×10^{-3}	32	30

Table 8: Training configuration, BERT-base across AG News, SST-2, QNLI, MRPC, RTE, and STS-B (Seesaw-v1).

Setting	**Value**
Backbone	`bert-base-uncased`
Target modules	Query projections only (12 layers)
Initial rank per module	8
Global rank budget R	96
Min. / Max. rank	2 / 16
Max. rank moved per transfer	1
LoRA α	16
Batch size	32
Max. sequence length	128
Learning rate	2×10^{-4}
Optimizer	AdamW (default $\beta_1, \beta_2, \epsilon$)
Epochs	5
Warm-up epochs (no controller)	1
Controller cadence	Every epoch after warm-up
Relative gap threshold τ	0.10
Random seed	42

B.4 Target Module Discovery and Initial Rank Allocation

For Seesaw-v2 runs, target Q/K/V/O modules are located programmatically rather than by hard-coded attribute names, since attention submodule naming is not guaranteed to be identical across Transformers library versions or DeBERTa variants. The discovery procedure inspects each layer's self-attention block for `nn.Linear` submodules, matches against a list of expected names (e.g., `query_proj`, `query`, `q_proj`), and falls back to positional ordering among discovered `nn.Linear` modules if no name match is found. The same fallback logic is applied to the output projection within each layer's attention-output block. This produces 48 target modules (12 layers $\times$ 4 projections) for both DeBERTa-v3-base and DeBERTa-v1-base, verified by assertion prior to LoRA insertion.

Given global rank budget R and M target modules, the initial per-module rank is assigned by integer division with the remainder distributed to the first modules encountered in layer-then-projection

Table 9: Training configuration, DeBERTa-v3-base across SST-2, CoLA, STS-B, and MRPC (Seesaw-v2, z-score).

Setting	Value
Backbone	`microsoft/deberta-v3-base`
Target modules	Query, Key, Value, Output (12 layers $\times$ 4 = 48)
Global rank budget R	827
Min. / Max. rank	2 / 64
LoRA α	32
Batch size (train / eval)	32 / 32
Max. sequence length	128
Learning rate	2×10^{-4}
Weight decay	0.01
Warm-up ratio	0.06
Gradient clipping (max norm)	1.0
Epochs	5
Controller start epoch	2
Controller interval	Every epoch
Importance EMA decay γ	0.85
Z-score gap threshold τ	0.25
Random seed	42

Table 10: Training configuration, DeBERTa-v1-base on SST-2 (Seesaw-v2, z-score).

Setting	Value
Backbone	`microsoft/deberta-base`
Target modules	Query, Key, Value, Output (12 layers $\times$ 4 = 48)
Global rank budget R	827
Min. / Max. rank	2 / 64
LoRA α	32
Batch size (train / eval)	32 / 32
Max. sequence length	128
Learning rate	2×10^{-4}
Weight decay	0.01
Warm-up ratio	0.06
Gradient clipping (max norm)	1.0
Epochs	5
Controller start epoch	2
Controller interval	Every epoch
Importance EMA decay γ	0.85
Z-score gap threshold τ	0.25
Random seed	42

order (layer 0 query, key, value, output; layer 1 query, ...):

$$r_l^{(0)} = \begin{cases} \lfloor R/M \rfloor + 1 & \text{if } l < (R \bmod M) \\ \lfloor R/M \rfloor & \text{otherwise,} \end{cases} \qquad \sum_{l=1}^{M} r_l^{(0)} = R. \tag{6}$$

For $R = 827$, $M = 48$: $\lfloor 827/48 \rfloor = 17$ with remainder $827 \bmod 48 = 11$, so 11 modules are initialized at rank 18 and the remaining 37 at rank 17 (Table 11). This initialization is identical across the DeBERTa-v3-base tasks (SST-2, CoLA, STS-B, MRPC) and the DeBERTa-v1-base run, since all share the same target-module ordering and global budget.

Table 11: Initial rank allocation across Q/K/V/O modules (DeBERTa-v3-base and DeBERTa-v1-base, $R = 827$, $M = 48$). Modules are enumerated in layer-then-projection order (query, key, value, output per layer).

Layer	Query	Key	Value	Output
0	18	18	18	18
1	18	18	18	18
2	18	18	18	17
3	17	17	17	17
4	17	17	17	17
5	17	17	17	17
6	17	17	17	17
7	17	17	17	17
8	17	17	17	17
9	17	17	17	17
10	17	17	17	17
11	17	17	17	17

Modules at rank 18: 11 Modules at rank 17: 37 Total rank: 827

Table 12: DeBERTa-v1 hyperparameters, SST-2.

Dataset	LR	Batch Size	Epochs
SST-2	2×10^{-4}	32	5

Table 13: DeBERTa-v3 hyperparameters.

Dataset	LR	Batch Size	Epochs
SST-2	2×10^{-4}	32	5
MRPC	1×10^{-3}	32	25
CoLA	2×10^{-4}	32	30
STS-B	1×10^{-4}	32	25