Bi-Phase Training: Learning Efficiently From Scratch

Abel Gurung
Purdue University
gurung1@purdue.edu

Joseph Campbell
Purdue University
joecamp@purdue.edu

Abstract

Pre-training foundational models is computationally expensive as models scale. In this work, we introduce Bi-Phase Training (BPT), a parameter-efficient pre-training method that preserves the high-rank learning dynamics observed in full-parameter optimization with substantially reduced trainable parameters. BPT parameterizes each weight update as a composition of constrained high-rank left/right transformations via trainable diagonal matrices and low-rank adapters whose updates are periodically merged into the frozen base weights, resulting in a compounding expansion of accessible directions during training. Across language-model pre-training from scratch (Qwen2.5 models from 100M to 1.5B parameters and OLMo-2-1B) on FineWeb-Edu (10B and 100B tokens), BPT achieves validation perplexity close to full-parameter baselines while using far fewer trainable parameters—e.g., at the 1.5B scale, BPT uses 66% fewer trainable parameters with only a small validation-perplexity gap. Additionally, a comprehensive evaluation across diverse downstream tasks shows that BPT preserves much of the full-parameterized model's downstream performance.

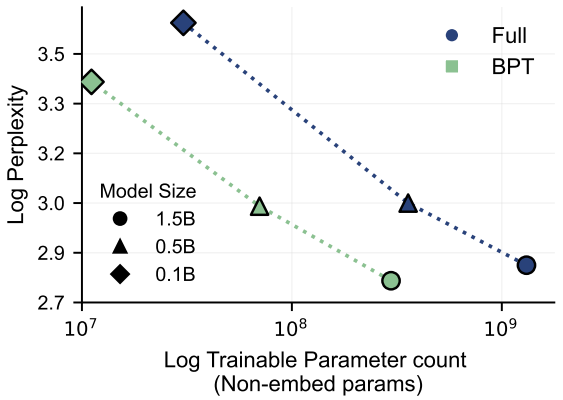

Figure 1: Evaluation log-perplexity vs. log trainable parameters (non-embedding) across model sizes. BPT approaches the fully-parameterized baseline while using far fewer trainable parameters; on the 1.5B model it achieves comparable eval loss with 66% fewer trainable parameters

1 Introduction

Foundational models have demonstrated impressive general-purpose performance (DeepSeek-AI et al., 2025; Grattafiori et al., 2024). These models usually consist of billions of parameters, are trained on massive datasets, and have become the standard in modern deep learning. Empirical studies such as scaling laws have demonstrated that increasing model size generally leads to lower training loss (Kaplan et al., 2020; Hoffmann et al., 2022). However, training larger models requires significantly more compute, highlighting the need for more compute-efficient pre-training methods.

During training, a given trainable parameter incurs a fixed memory overhead: the parameter itself (n), its gradient (n), and two optimizer states ($2n$) when using Adam family of optimizers (Kingma and Ba, 2017; Loshchilov and Hutter, 2019). While total training memory depends on several factors such as batch size and activation footprint (Chen et al., 2016), this $4n$ overhead remains fixed per trainable parameter. Distinguishing between non-trainable and trainable parameters for pre-training is important, and the total number of parameters can be interpreted as the model's representational capacity. Our central claim is that effective pre-training does not require optimizing every parameter. Instead, models can retain much of the learning behavior of full training while updating only a small subset of parameters, provided that the resulting updates preserve the essential training dynamics. We first analyze the general learning dynamics of full-parameter training in terms of the rank of the update for SGD and Adam optimizers, showing that in full-parameter training, updates are high-rank, and verify this behavior on a simple

model. Then, to preserve the learning dynamics of high-rank while being parameter-efficient, we introduce Bi-Phase Training (BPT), a novel parameter-efficient method for pre-training foundation models that updates a weight matrix W by simultaneously applying constrained high-rank and low-rank updates. This work directly builds on prior works such as LoRA (Hu et al., 2021), ReLoRA (Lialin et al., 2023), and HyperAdapt (Gurung and Campbell, 2025) and combines them in pre-training setting to achieve high-rank updates at every optimization step.

The claim that a high-rank update is necessary is consistent with prior evidence. ReLoRA (Lialin et al., 2023) shows that allowing a brief period of full-parameter training before switching to low-rank adaptation improves optimization, suggesting early learning benefits from richer (higher-rank) update directions (Lialin et al., 2023). Complementarily, systematic ablations that increase LoRA rank r toward full-rank updates report that doing this matches full training performance when the model is trained from scratch, reinforcing the link between update rank and training performance (Huh et al., 2024).

Empirically, we demonstrate the effectiveness of our method by pre-training three language models of varying sizes, showing consistent performance across scales. We also perform a comprehensive downstream evaluation of the trained models to show that there is minimal performance loss when using fewer trainable parameters than when using all trainable parameters. Our main contributions are as follows:

- We introduce Bi-Phase Training (BPT) to significantly reduce the number of trainable parameters required during pre-training.
- We provide a theoretical upper-bound to the rank of the update induced by BPT.
- We comprehensively test the method on downstream evaluation tasks and show that there is minimal performance difference between the fully parameterized training and BPT.

2 Background

Full Parameterized Training During neural network training, we aim to find weight matrices $\mathrm{W} \in \mathbb{R}^{n \times m}$ that minimizes a loss function, L. The matrix is updated iteratively using gradient signals from backpropagation. The rank of the update $\Delta\mathrm{W}$ indicates the number of independent update directions applied per step to the weight matrix during optimization. Formally, we want to optimize the weight matrix W using $\Delta\mathrm{W}$: $\mathrm{W}' = \mathrm{W} + \Delta\mathrm{W}$ where W' is the optimized weight matrix. For a single training example with an input vector $x \in \mathbb{R}^m$ and a corresponding layer output $y \in \mathbb{R}^n$, the gradient of the loss with respect to the weight matrix G is the outer product of the upstream gradient and the input vector's transpose:

$$G = \frac{\partial L}{\partial \mathrm{W}} = \left(\frac{\partial L}{\partial y}\right) x^T$$

Since this is an outer product of two vectors, its rank is at most one, meaning that one training data point gives rank-one information. In stochastic gradient descent (SGD), where only the gradient of the matrix is used for the update, $\Delta\mathrm{W}$, the rank of $\Delta\mathrm{W}$ for a single example is at most one. For a mini-batch of B training examples, we can stack the inputs into a matrix $\mathrm{X} \in \mathbb{R}^{m \times B}$ and the corresponding upstream gradients into a matrix $\Lambda \in \mathbb{R}^{n \times B}$. The gradient for the entire mini-batch is then given by the matrix product:

$$G = \frac{1}{B}\Lambda\mathrm{X}^T = \frac{1}{B}\sum_{i=1}^{B} \lambda_i x_i^T$$

The rank of a matrix product is less than or equal to the minimum rank of its factors. Since the rank of Λ is at most $\min(n, B)$ and the rank of X^T is at most $\min(B, m)$, the rank of the resulting gradient matrix is bounded by:

$$\mathrm{rank}(G) \leq \min(\mathrm{rank}(\Lambda), \mathrm{rank}(\mathrm{X}^T))$$

Without loss of generality, let $B \leq \min(n, m)$. For the update $\Delta\mathrm{W}$ in SGD, the $\mathrm{rank}(\Delta\mathrm{W})$ is at most B. This means that the update rank's upper-bound grows linearly with the batch size for SGD. Additionally, adding momentum or using Adam has a different upper-bound for the update rank even at smaller batch size, refer to Appendix A for more on momentum and Adam's rank.

Parameter Efficient Training Full parameterized training can induce high-rank updates. However, this requires a large number of trainable parameters which is computationally expensive. In this section, we introduce parameter-efficient methods to train models while reducing the number of trainable parameters. To reduce the number of trainable parameters in W, we want to express $\Delta\mathrm{W}$

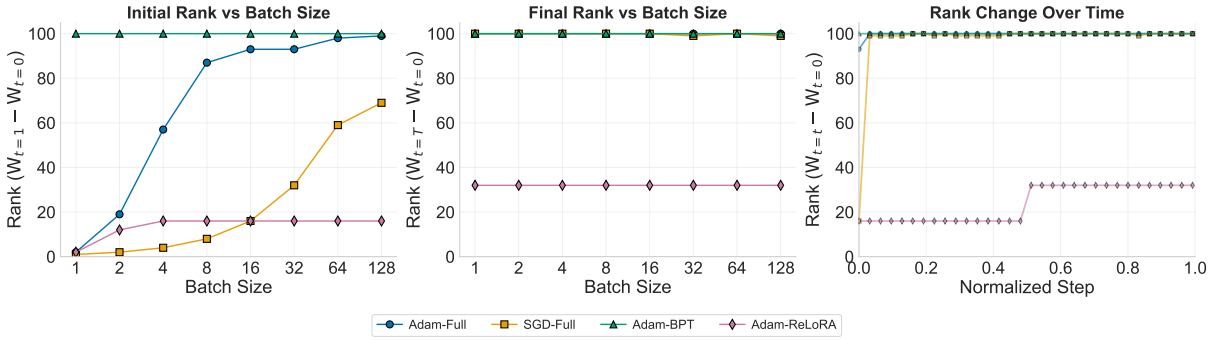

Figure 2: Update-rank trend across batch sizes and methods. Left: rank change after the first update step versus batch size, showing scaling effects for SGD and consistently high-rank updates for Adam-full and BPT. Middle: rank at the end of training ($t = T$) versus batch size, where full-parameter methods and BPT reach near-full rank even from small batches, while ReLoRA remains rank-limited. Right: rank evolution over normalized training steps for batch size 16, comparing Adam-full, SGD-full, BPT, and ReLoRA.

with minimal number of trainable parameters while having high-rank such that the trainable parameter reduction does not alter its learning dynamics.

In LoRA (Hu et al., 2021), $\Delta\mathrm{W}$ is defined as the product of two low-rank matrices $\mathrm{B} \in \mathbb{R}^{n\times r}$ and $\mathrm{A} \in \mathbb{R}^{r\times m}$ such that $\Delta\mathrm{W} = \alpha\mathrm{BA}$, where α is a scaling factor. LoRA trains the low-rank matrices, B and A, which contain significantly fewer trainable parameters than W. LoRA yields strong results for fine-tuning a pre-trained weight matrix, however, the objective landscape for pre-training from scratch is too complex to be modeled by a single r-rank subspace alone (Lialin et al., 2023). Previous works (Huh et al., 2024) have empirically shown that for training models from scratch with LoRA with rank $r = \min(n, m)$ equaling the maximum rank possible of the matrix can match the performance of full training from scratch. Since the $\Delta\mathrm{W}$ rank in LoRA is always bounded by r, it can never utilize the full rank potential of W. To address this limitation, ReLoRA (Lialin et al., 2023) extends LoRA by accumulating low-rank updates over multiple steps:

$$\Delta\mathrm{W} = \alpha \sum_{n=1}^{N} \mathrm{B}_n\mathrm{A}_n$$

where B and A are low-rank matrices. During training, B and A are optimized, and their low-rank matrices are periodically merged back into the original weight matrix W. Following each merge, B and A are reinitialized, allowing the accumulated update $\Delta\mathrm{W}$ to surpass the rank constraint r after multiple merge cycles. Despite this improvement, the rank of updates at **any single step** remains confined to a low-rank subspace. Also, for the duration of Δt between two merge cycles, the update is restricted to only a rank-r update. This limits its ability to learn as efficiently as a fully parameterized model, as the update rank for a full parameterized model for any given timestep can be high-rank and results in weaker performance compared to full parameter pre-training (Lialin et al., 2023). Additionally, merging less frequently gives better performance empirically, since the Adam EMAs are better estimates as training progresses (Lialin et al., 2023; Huh et al., 2024).

In a related work, HyperAdapt (Gurung and Campbell, 2025) proposed a parameter-efficient high-rank adaptation method by defining the update as: $\Delta\mathrm{W} = \mathrm{AWB} - \mathrm{W}$ where $\mathrm{A} \in \mathbb{R}^{n\times n}$ and $\mathrm{B} \in \mathbb{R}^{m\times m}$ are trainable diagonal matrices, and W is non-trainable parameter. Since both trainable matrices are diagonal, only $n + m$ number of parameters are needed to train them. Here the rank of $\Delta\mathrm{W}$ is upper-bounded by the $2 \cdot \mathrm{rank}(\mathrm{W})$. However, this method assumes that pre-trained weight matrices already contain relevant orthogonal directions which can be scaled relevant to the downstream fine-tuning task, strictly limiting its application to fine-tuning pre-trained models.

Empirical rank. We train a three-layer ReLU MLP on CIFAR-10 (Torralba et al., 2008) and track the second-layer update rank, $\mathrm{rank}(W_t - W_0)$, across batch sizes and training steps; results are shown in Figure 2. After one update, Adam-full and BPT produce consistently high-rank updates across batch sizes, while SGD-full follows its batch-size-dependent rank bound and ReLoRA remains rank-limited by construction. This suggests that BPT's diagonal matrices can efficiently mod-

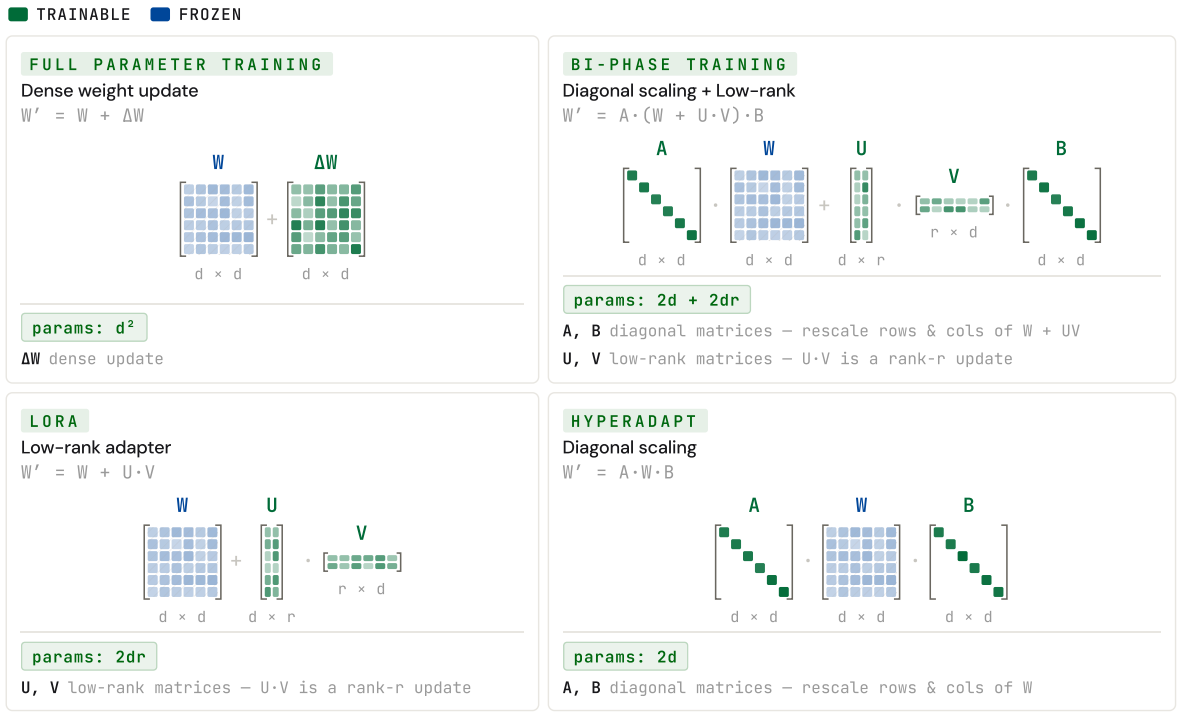

Figure 3: Overview of parameter-efficient training techniques. Blue represents non-trainable parameters, and green represents trainable parameters. UV is the product of two low-rank matrices of rank-r, and A and B are diagonal matrices.

ulate the rank-one directions of the base weight matrix. By the end of training, all methods except ReLoRA reach near-full-rank updates regardless of batch size, with BPT closely matching full-parameterized training. For batch size 16, SGD-full begins with low-rank updates but reaches full rank within a few steps, indicating that high-rank updates emerge quickly during full training.

3 Our Method

Since a fully parameterized model can express high-rank updates at any step t, as shown in Figure 2, we seek an update $\Delta\mathrm{W}$ whose rank is comparably high while minimizing total number of trainable parameter to express $\Delta\mathrm{W}$. To do this, we introduce Bi-Phase Training (BPT), a parameter-efficient method designed to induce high-rank updates within individual weight matrices using diagonal matrices and build low-rank subspace using low-rank matrices.

Bi-Phase Training Our method conceptualizes the optimization of a weight matrix $\mathrm{W} \in \mathbb{R}^{n\times m}$ not as a direct update to all $n \times m$ parameters, but as an update structure $\Delta\mathrm{W}$ that combines transformations applied through efficient high-rank diagonal matrices with exploration guided by low-rank matrices. This allows us to approximate the expressiveness of full parameter updates while drastically reducing the trainable parameter count. Formally, for a weight matrix W_0, the update $\Delta\mathrm{W}$ is defined as:

$$\Delta\mathrm{W} = \mathrm{A}\left(\mathrm{W}_0 + \sum_{t=1}^{T}\mathrm{UV}\right)\mathrm{B} - \mathrm{W}_0 \qquad (1)$$

where $\mathrm{A} \in \mathbb{R}^{n\times n}$ and $\mathrm{B} \in \mathbb{R}^{m\times m}$ are diagonal trainable matrices, while $\mathrm{U} \in \mathbb{R}^{n\times r}$ and $\mathrm{V} \in \mathbb{R}^{r\times m}$ are low-rank trainable matrices. Importantly, W_0 is a non-trainable parameter during training. At a given time t, we only keep a pair of U and V trainable and periodically merge U and V to W. So that means for a given step t, we have:

$$\Delta\mathrm{W} = \mathrm{A}\left(\mathrm{W}_0 + \mathrm{UV}\right)\mathrm{B} - \mathrm{W}_0 \qquad (2)$$

At any given time t, the update in Equation 2 is not restricted to rank r in the same way as a pure low-rank adapter. The lemma below provides an expressivity upper bound, $\mathrm{rank}(\Delta\mathrm{W}) \leq \min(n, m, 2R + r)$, showing that BPT's update rank is not inherently limited by the rank of UV. Intuitively, the low-rank matrices U and V find and accumulate relevant orthogonal directions within

a lower-dimensional subspace, while the diagonal matrices A and B scale relevant orthogonal directions, thus effectively guiding the high-rank updates.

Lemma 3.1. *The upper bound for the rank of the update* $\Delta \mathrm{W} = \mathrm{A}(\mathrm{W} + \mathrm{UV})\mathrm{B} - \mathrm{W}$ *is* $2R + r$*, where* $R = \mathrm{rank}(\mathrm{W})$ *and* $r = \mathrm{rank}(\mathrm{UV})$.

The proof is in the appendix Appendix B.

3.1 Mechanism of Learning

BPT combines two complementary mechanisms: low-rank exploration and diagonal reweighting. Let the frozen base matrix be decomposed into rank-one directions,

$$\mathrm{W}_0 = \sum_{i=1}^{R} \mathbf{a}_i \mathbf{b}_i^\top.$$

The diagonal matrices A and B rescale these existing directions:

$$\mathrm{A}\mathrm{W}_0\mathrm{B} = \sum_{i=1}^{R} \mathrm{A}\mathbf{a}_i \mathbf{b}_i^\top \mathrm{B}.$$

However, diagonal scaling alone cannot create new directions. The low-rank term UV provides this exploration:

$$\mathrm{A}(\mathrm{W}_0 + \mathrm{UV})\mathrm{B} = \sum_{i=1}^{R} \mathrm{A}\mathbf{a}_i \mathbf{b}_i^\top \mathrm{B} + \sum_{j=1}^{r} \mathrm{A}\mathbf{u}_j \mathbf{v}_j^\top \mathrm{B}.$$

Thus, U and V discover new low-rank directions, while A and B immediately modulate both the original and newly discovered directions. Periodically, BPT merges the learned low-rank update into the base matrix,

$$\mathrm{W}_{t+1} := \mathrm{W}_t + \mathrm{U}_t\mathrm{V}_t, \tag{3}$$

and reinitializes U and V to explore new subspaces. Since the diagonal matrices are not merged or reinitialized, they continue to rescale all accumulated directions. After multiple merge cycles, BPT therefore compounds learning: low-rank factors add new directions to W, while the diagonals continuously scale both old and new directions. This differs from ReLoRA-style training, where each update remains confined to a rank-r subspace between merge steps. In BPT, the current low-rank factors explore new directions while the persistent diagonal matrices simultaneously reshape the accumulated base, allowing high-rank behavior at each step.

3.2 Initialization and Merging

Initialization We initialize the fixed base weight W with Kaiming initialization (He et al., 2015). The diagonal matrices A and B are initialized to one, so they initially act as identities. For the low-rank factors, U is initialized semi-orthogonally and V is initialized to zero. Thus $\mathrm{UV} = 0$, and the initial forward pass matches the Kaiming-initialized base model.

Merging During training, BPT periodically adds the learned low-rank update into the frozen base weight using Equation 3. This preserves the directions already discovered while freeing the low-rank factors to explore a new subspace. After each merge, we reset U and V as

$$\mathrm{U} := \text{init.orthogonal} \quad , \quad \mathrm{V} := \text{init.zero}$$

Because $\mathrm{UV} = 0$ immediately after reset, the effective matrix is unchanged at the merge point. The fresh semi-orthogonal U provides rank-r directions for the next exploration phase. We do not merge the diagonal matrices, since doing so would not create new subspace directions.

Reinitializing the low-rank factors makes their optimizer exponential moving averages (EMAs) stale, whereas the diagonal EMAs remain valid because the diagonals are not reset. This helps stabilize training and avoids sudden loss spikes after merging.

3.3 Parameter and Compute Efficiency

The primary advantage of BPT lies in its significant reduction in the number of trainable parameters compared to training the full weight matrix W. For a single layer with weight matrix $\mathrm{W} \in \mathbb{R}^{n \times m}$, the trainable parameters in BPT are located only in the diagonal matrices A and B, and the low-rank matrices U and V. The total number of trainable parameters for one such layer in BPT is $n + m$ (from diagonal A and B) plus $n \times r + r \times m$ (from low-rank U and V). In contrast, training the full weight matrix requires $n \times m$ parameters. Since $r \ll \min(n, m)$, the total trainable parameters in BPT are substantially less than nm.

$$\underbrace{n+m}_{\text{Diagonal Matrices}} + \underbrace{nr+rm}_{\text{Low-Rank Matrices}} \ll \underbrace{nm}_{\text{Full Matrix}}$$

This parameter reduction directly translates into lower memory overhead during training with efficient kernel implementation. This trainable pa-

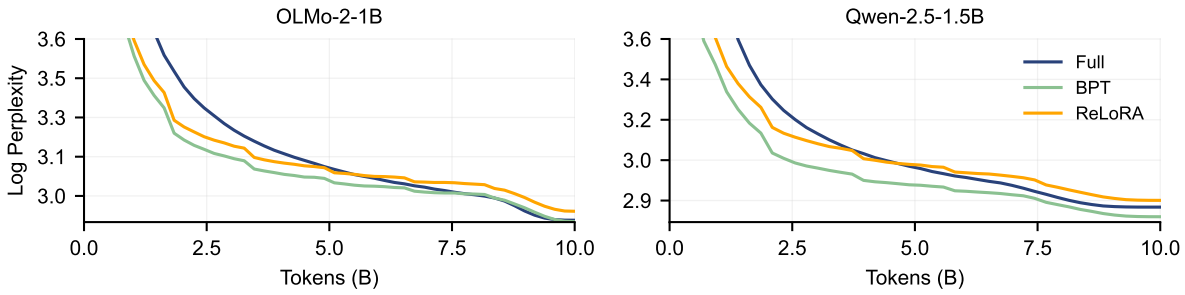

Figure 4: Eval loss over training steps. Left: OLMo-2-1B. Right: Qwen-2.5-1.5B. BPT tracks the Full baseline closely and outperforms ReLoRA over time, despite using substantially fewer trainable parameters.

rameter count reduces the fixed memory cost associated with training neural networks as stated in section 1. Additionally, BPT requires approximately two-thirds of the linear-layer FLOPs of full training; we provide the full FLOP analysis in Appendix C.

4 Related Work

While the past few years have witnessed remarkable progress in reducing the memory footprint for fine-tuning large language models, memory-efficient pre-training remains considerably less explored. GaLore (Zhao et al., 2024) addresses this gap by proposing a novel approach to parameter-efficient training that projects gradients into a low-rank subspace, diverging from traditional methods that directly parameterize low-rank weight matrices. However, GaLore's reliance on Singular Value Decomposition (SVD) to identify the optimal low-rank approximation presents a significant computational bottleneck. The computational cost of SVD scales poorly with the matrix's dimensions, since SVD has $O(n^3)$ time complexity, making GaLore difficult to scale.

LoRA-the-explorer (LTE) (Huh et al., 2024) extends LoRA to pre-training by exploring the low-rank solution space in parallel across different computing nodes. While LTE's parallel search strategy presents a seemingly promising avenue to mitigate the computational bottlenecks associated with full-rank pre-training, it unfortunately inherits the same fundamental drawback: at each step, optimization is limited to a low-rank subspace similar to other low-rank methods section 2.

Other LoRA variants further improve parameter-efficient adaptation but remain primarily designed for fine-tuning pretrained models. DoRA (Liu et al., 2024) decomposes each pretrained weight into magnitude and direction components, using low-rank weights for directional updates while learning the magnitude separately. VeRA (Kopiczko et al., 2024) instead freezes shared random low-rank matrices and trains only lightweight scaling vectors, greatly reducing adapter parameters. However, both approaches still rely on constrained low-rank directions or fixed random projections, and therefore do not directly address the need for repeatedly expanding high-rank update directions during from-scratch pre-training.

5 Empirical Experiments

To evaluate the effectiveness of our proposed method, we conducted a series of pre-training experiments on language models of varying scales and assessed their performance on a diverse suite of downstream tasks. Our primary goal was to demonstrate that BPT can come close to the performance of standard full-parameter pre-training while requiring significantly fewer trainable parameters, thereby lowering the computational barrier to developing large foundation models.

All models were pre-trained from **scratch** using the standard next-token prediction objective. We used a 10 and 100 billion token subset of the FineWeb-Edu dataset (Penedo et al., 2024). Our baselines for comparison are standard full-parameter pre-training (referred to as "Full"), where all model parameters are trainable, and ReLoRA. Importantly, all methods use the same underlying model architecture and the same total number of parameters; the difference lies solely in the number of trainable parameters during training. We train three scales of models based on the Qwen2.5 architecture: 100M, 0.5B, and 1.5B parameters. Additionally, we also pre-train OLMo-2-1B (OLMo et al., 2024) on the 10B tokens of the same dataset. The detailed specifications of the model architectures are provided in Ap-

Model	Method	Trainable Params	Wiki ppl ↓	LAMB ppl ↓	LAMB acc ↑	Hella acc ↑	WinoG acc ↑	PIQA acc ↑	BoolQ acc ↑	SciQ acc ↑	OBQA acc ↑	ARC-E acc ↑	ARC-C acc ↑	**Avg**	SWDE acc ↑
10 Billion Tokens															
Qwen-2.5-1.5B	Full	1.5B	2.0	47.9	30.9	33.1	51.3	66.9	59.9	81.1	23.6	59.2	23.0	47.7	17.7
	ReLoRA	0.5B	5.0	42.8	32.2	33.1	50.2	66.5	60.6	82.6	23.2	58.8	24.5	48.0	15.0
	BPT	0.5B	4.8	34.6	34.6	34.7	52.1	66.8	57.2	81.2	23.0	61.9	27.3	48.8	15.3
OLMo-2-1B	Full	1.5B	1.9	48.2	30.6	32.8	51.1	66.6	57.9	82.8	23.2	58.9	23.5	47.5	14.4
	ReLoRA	0.6B	2.0	47.9	30.9	32.8	52.2	66.5	57.9	83.7	23.4	58.8	25.4	48.0	16.5
	BPT	0.6B	2.0	40.7	33.0	33.1	52.6	67.7	60.7	82.2	23.6	59.9	25.7	48.7	19.5
100 Billion Tokens															
Qwen-2.5-1.5B	Full	1.5B	3.4	18.1	42.8	41.4	53.5	71.6	59.7	87.5	28.8	69.2	34.0	54.3	23.9
	ReLoRA	0.5B	3.9	23.5	38.7	37.6	53.3	68.4	58.6	84.4	25.4	65.2	30.6	51.3	21.4
	BPT	0.5B	3.7	21.4	39.2	39.2	53.9	70.6	62.0	87.5	26.2	66.1	30.6	52.8	19.7

Table 1: Performance across Language Modeling, Commonsense, QA, and Extraction Tasks: lower is better for perplexity; higher is better for accuracy.

pendix D. We also summarize the parameter reduction achieved through BPT in Table 4. For more details regarding hyperparameters and layers targeted, refer to Appendix E. As the model scales, BPT uses a smaller fraction of trainable parameters while remaining close to full-parameter performance, since layers like the embedding layer do not have an outsized effect on larger models

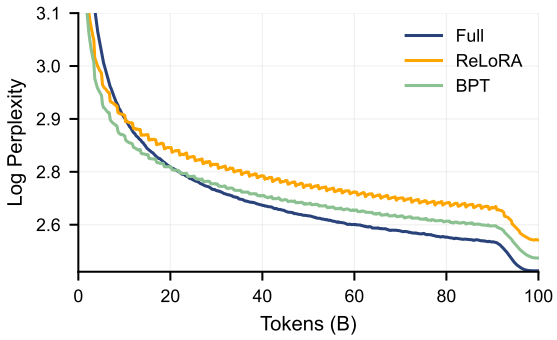

Figure 5: Qwen-2.5-1.5B: Validation Loss vs. Training Steps (Full vs. BPT) for 100B tokens

5.1 Pre-training Performance

Our pre-training experiments at 10 billion tokens demonstrate BPT achieves validation perplexity close to full-parameter training across different model scales while achieving substantial reductions in trainable parameters. Figure 1 summarizes our experimental results across different model sizes. The x-axis represents the trainable parameters, not including embedding parameters. We show that, for a given model size, BPT achieves log-perplexity performance on a validation set similar to that of the full parameterized model, with significantly fewer trainable parameters.

Figure 4 compares the evaluation log perplexity on the validation set for the Qwen-2.5-1.5B and OLMo-2 between the Full, ReLoRA, and BPT. We show that even with **66% less trainable parameters**, BPT achieves validation loss close to full-parameter training. The log-perplexity of validation data shows that the model does not simply overfit the training data and has a similar evaluation loss profile as the baseline. Both BPT and ReLoRA have almost an identical number of trainable parameters. ReLoRA's eval loss slowly diverges from BPT's, even though both methods use a similar high learning rate.

Longer Training (100 Billion Tokens) To further assess the effectiveness of BPT, we pre-train Qwen-2.5-1.5B model on 100 billion tokens from FineWeb-Edu using the same setup as our main experiments. The final evaluation difference between BPT and Full is **0.0271 log perplexity** with 66% fewer trainable parameters, compared with a 0.0656 gap between ReLoRA and Full.

5.2 Downstream Evaluation

The training loss and evaluation loss indicate strong performance for our proposed methods. To further evaluate whether the parameter efficiency of BPT impacts the model's ability to generalize to unseen tasks, we perform downstream evaluations on our Qwen-2.5-1.5B and OLMo-2-1B models. For all our evaluations, we do not further finetune any additional dataset and report zero-shot performance. Evaluations were performed using the LM evaluation harness (Gao et al., 2024). Following prior work at a similar model and data scale (Allen-Zhu, 2025; Yang et al., 2025), we evaluate commonsense and reasoning benchmarks including HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2019), PIQA (Bisk et al., 2019), and BoolQ (Clark

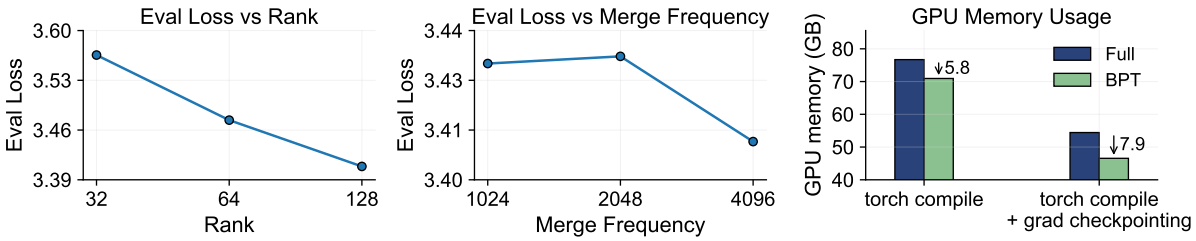

Figure 6: BPT ablations and memory summary. Left/Middle: rank and merge-frequency ablations on the 100M model trained on 10B tokens. Higher rank for the low-rank matrices leads to lower validation loss. Less frequent merges also lead to lower validation loss. Right: Peak GPU memory for Qwen-2.5-1.5B. BPT saves 5.8 GB to train a 1.5B model.

et al., 2019). We also report perplexity on held-out text using WikiText and LAMBADA (Paperno et al., 2016), and evaluate generation using SWDE (Lockard et al., 2019). For Wikitext, we report byte perplexity. Table 1 summarizes our results. At 10B tokens, BPT is competitive with Full on average in our benchmark suite, while at 100B tokens Full performs better on average. This suggests that BPT preserves much, but not all, downstream performance under reduced trainable-parameter training.

5.3 Comparison of PEFT Methods

Table 2: Comparison of PEFT Methods: Trainable Parameter and Eval Loss. Models are pre-trained from scratch on 10 billion tokens.

Method	Trainable Param	Eval Loss ↓
HyperAdapt	137M	5.486
VeRA	137M	4.640
LoRA	206M	3.051
DoRA	206M	3.102
ReLoRA	206M	3.038
BPT	206M	3.014
Full	494M	3.022

Since BPT builds on prior works such as LoRA, ReLoRA, and HyperAdapt, we can evaluate the performance of each of these components by directly pre-training using these methods and comparing against BPT. For this ablation study, we pre-train the Qwen-2.5-0.5B model in the same setup as our main experiments with 10 billion tokens and report their evaluation loss along with the trainable parameters. This study also allows us to see the effects of only training low-rank matrices UV without merging (LoRA), the effect of training low-rank matrices UV with merging (ReLoRA), and the effect of training only the diagonal matrices A and B (HyperAdapt).

The result is summarized in Table 2. HyperAdapt, which only uses diagonal matrices to update W, performs poorly since the base weight matrix W does not contain any relevant subspace that can be adjusted, since W at initialization is just a random matrix. LoRA without periodic rank accumulation performs better than HyperAdapt, and the difference between LoRA's evaluation loss and ReLoRA's evaluation loss is only 0.01. BPT and Full both have evaluation losses that are similar compared to other methods.

5.4 Rank and Merge Frequency

To better understand the role of rank and merge frequency in BPT, we ablate both under the same setting as our main results, using the 100M model trained on 10B tokens. As shown in Figure 6, increasing rank improves evaluation loss with diminishing returns, while less frequent merges generally perform better by allowing Adam to estimate its moments more stably between resets. Detailed GPU memory measurements are provided in Appendix F.

6 Conclusion

In this paper, we introduced Bi-Phase Training (BPT), a parameter-efficient pre-training method that combines diagonal high-rank transformations with periodically merged low-rank adapters. Across language-model pre-training from 100M to 1.5B parameters, BPT closely tracks full-parameter training while using substantially fewer trainable parameters, including 66% fewer at 1.5B. Downstream evaluations show minimal degradation, suggesting BPT as a practical route to more efficient foundation-model pre-training.

Limitations

Our experiments are limited to autoregressive language-model pre-training models up to 1.5B parameters. Future work should test BPT on larger models, longer training runs, other architectures and also different modalities such as diffusion.

References

Zeyuan Allen-Zhu. 2025. Physics of language models: Part 4.1, architecture design and the magic of canon layers. *Preprint*, arXiv:2512.17351.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. Piqa: Reasoning about physical commonsense in natural language. *Preprint*, arXiv:1911.11641.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *Preprint*, arXiv:1604.06174.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *Preprint*, arXiv:1905.10044.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. The language model evaluation harness.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Abel Gurung and Joseph Campbell. 2025. Hyperadapt: Simple high-rank adaptation. *Preprint*, arXiv:2509.18629.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Preprint*, arXiv:1502.01852.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, and 3 others. 2022. Training compute-optimal large language models. *Preprint*, arXiv:2203.15556.

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

Minyoung Huh, Brian Cheung, Jeremy Bernstein, Phillip Isola, and Pulkit Agrawal. 2024. Training neural networks from scratch with parallel low-rank adapters. *Preprint*, arXiv:2402.16828.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *Preprint*, arXiv:2001.08361.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization. *Preprint*, arXiv:1412.6980.

Dawid J. Kopiczko, Tijmen Blankevoort, and Yuki M. Asano. 2024. Vera: Vector-based random matrix adaptation. *Preprint*, arXiv:2310.11454.

Vladislav Lialin, Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky. 2023. Relora: High-rank training through low-rank updates. *Preprint*, arXiv:2307.05695.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation. *Preprint*, arXiv:2402.09353.

Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. 2019. OpenCeres: When open information extraction meets the semi-structured web. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3047–3056, Minneapolis, Minnesota. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. *Preprint*, arXiv:1711.05101.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, and 21 others. 2024. 2 OLMo 2 Furious. *Preprint*, arXiv:2501.00656.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The lambada dataset: Word prediction requiring a broad discourse context. *Preprint*, arXiv:1606.06031.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. *Preprint*, arXiv:2406.17557.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Winogrande: An adversarial winograd schema challenge at scale. *Preprint*, arXiv:1907.10641.

John Schulman and Thinking Machines Lab. 2025. Lora without regret. *Thinking Machines Lab: Connectionism*. Https://thinkingmachines.ai/blog/lora/.

Antonio Torralba, Rob Fergus, and William T. Freeman. 2008. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. 2025. Parallelizing linear transformers with the delta rule over sequence length. *Preprint*, arXiv:2406.06484.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *Preprint*, arXiv:1905.07830.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. 2024. Galore: Memory-efficient llm training by gradient low-rank projection. *Preprint*, arXiv:2403.03507.

A Momentum and Adam's Update Rank

For SGD with momentum, the update rank at time t is upper-bounded by tB since momentum is a linear combination of past gradients. In modern optimizers such as Adam (Kingma and Ba, 2017), where first and second exponential moving averages (EMAs) are stored and transformed, the rank of the update is no longer bounded just by the batch size but depends on the rank of its first and second EMAs. Here we derive a loose upper-bound for Adam.

For Adam, let $G_t \in \mathbb{R}^{n\times m}$ denote the mini-batch gradient at step t (for batch size B, we have $\operatorname{rank}(G_t) \le \min(n, m, B)$ as above). Adam maintains exponential moving averages of the first and second moments:

$$M_t = \beta_1 M_{t-1} + (1-\beta_1)G_t,$$
$$V_t = \beta_2 V_{t-1} + (1-\beta_2)(G_t \odot G_t),$$

where $\odot$ denotes the element-wise product. The parameter update is

$$\Delta W_t = -\alpha \frac{M_t}{\sqrt{V_t}+\varepsilon} = -\alpha M_t \odot \frac{1}{\sqrt{V_t}+\varepsilon}$$

where the division and square-root are applied element-wise. Using Hadmard product

Since M_t is a linear combination of past gradients,

$$M_t = (1-\beta_1)\sum_{k=1}^{t} \beta_1^{t-k} G_k,$$

and using $\operatorname{rank}(\sum_{k=1}^{t} G_k) \le \sum_{k=1}^{t} \operatorname{rank}(G_k)$, we obtain an upper-bound

$$\operatorname{rank}(M_t) \le \sum_{k=1}^{t} \operatorname{rank}(G_k) \le tB,$$

hence $\operatorname{rank}(M_t) \le \min(n, m, tB)$. Using the Hadamard-rank inequality: for any $A, B \in \mathbb{R}^{n\times m}$,

$$\operatorname{rank}(A \odot B) \le \operatorname{rank}(A)\ \operatorname{rank}(B).$$

Applying this to ΔW_t yields

$$\operatorname{rank}(\Delta W_t) = \operatorname{rank}(M_t \odot \frac{1}{\sqrt{V_t}+\varepsilon}) \le \operatorname{rank}(M_t)\ \operatorname{rank}(\frac{1}{\sqrt{V_t}+\varepsilon})$$

Hence, for Adam, the rank of its ΔW_t is upper-bounded by the product of its first and second moments' ranks. Here, we do not provide the rank of the second moment primarily because performing element-wise square-root on a matrix leads to full rank in some cases, even for rank-one matrices.

B BPT Rank Proof

Lemma B.1. *The upper bound for the rank of the update* $\Delta\mathrm{W} = \mathrm{A}(\mathrm{W}+\mathrm{UV})\mathrm{B} - \mathrm{W}$ *is* $2R + r$*, where* $R = \operatorname{rank}(\mathrm{W})$ *and* $r = \operatorname{rank}(\mathrm{UV})$.

Proof. We have

$$\Delta\mathrm{W} = \mathrm{A}(\mathrm{W}+\mathrm{UV})\mathrm{B} - \mathrm{W} = \mathrm{AWB} + \mathrm{AUVB} - \mathrm{W}.$$

For all conformable matrices X and Y, $\operatorname{rank}(\mathrm{X}+\mathrm{Y}) \le \operatorname{rank}(\mathrm{X}) + \operatorname{rank}(\mathrm{Y})$. Therefore,

$$\operatorname{rank}\big(\mathrm{AWB} + \mathrm{AUVB} - \mathrm{W}\big) \le \operatorname{rank}(\mathrm{AWB}) + \operatorname{rank}(\mathrm{AUVB}) + \operatorname{rank}(-\mathrm{W}).$$

Since matrix multiplication cannot increase rank, $\operatorname{rank}(\mathrm{AXB}) \le \operatorname{rank}(\mathrm{X})$ for any X, and hence

$$\operatorname{rank}\big(\mathrm{AWB} + \mathrm{AUVB} - \mathrm{W}\big) \le \operatorname{rank}(\mathrm{W}) + \operatorname{rank}(\mathrm{UV}) + \operatorname{rank}(-\mathrm{W}).$$

We have $\operatorname{rank}(\mathrm{UV}) = r$ and $\operatorname{rank}(\mathrm{W}) = \operatorname{rank}(-\mathrm{W}) = R$, so

$$\operatorname{rank}\big(\Delta\mathrm{W}\big) \le R + r + R = 2R + r.$$

Hence the rank of the update $\Delta\mathrm{W}$ is upper-bounded by $2R + r$. Trivially, it is also bounded by its dimensions so $\operatorname{rank}(\Delta\mathrm{W}) \le \min(n, m, 2R + r)$ □

C FLOP Analysis

For FLOP analysis, we use the same setup as (Schulman and Lab, 2025). In full training, for a weight matrix $\mathrm{W} \in \mathbb{R}^{N \times N}$, an input $x \in \mathbb{R}^N$, and an output $y \in \mathbb{R}^N$, we have

$$\begin{aligned} \text{Forward:} \quad & y = \mathrm{W}x && \left(N^2 \text{ multiply-adds}\right), \\ \text{Backward:} \quad & \bar{x} = \mathrm{W}^\top \bar{y} && \left(N^2 \text{ multiply-adds}\right), \\ & \bar{\mathrm{W}} \mathrel{+}= \bar{y} x^\top && \left(N^2 \text{ multiply-adds}\right). \end{aligned}$$

Here, $\bar{\mathrm{W}}$, $\bar{y}$, and $\bar{x}$ denote gradients of the loss with respect to W, y, and x, respectively. Combining the forward pass $\left(N^2\right)$ and backward pass $\left(2N^2\right)$ gives

$$\mathrm{FLOPs}_{\mathrm{Full}} = 3N^2.$$

In BPT, we replace W by $\mathrm{A}(\mathrm{W} + \mathrm{UV})\mathrm{B}$, where $\mathrm{A}, \mathrm{B} \in \mathbb{R}^{N \times N}$ are diagonal matrices and $\mathrm{U} \in \mathbb{R}^{N \times R}$, $\mathrm{V} \in \mathbb{R}^{R \times N}$ are low-rank matrices with $R \ll N$. Since we only update $\bar{\mathrm{A}}$, $\bar{\mathrm{B}}$, $\bar{\mathrm{U}}$, and $\bar{\mathrm{V}}$, the update of $\bar{\mathrm{W}}$ is replaced by cheaper operations.

The FLOP estimate for BPT is therefore

$$\begin{aligned} \text{Forward:} \quad & y = \mathrm{A}(\mathrm{W} + \mathrm{UV})\mathrm{B}x && \left(N^2 + 2NR + 2N \text{ multiply-adds}\right), \\ \text{Backward:} \quad & \bar{x} = \mathrm{B}^\top(\mathrm{W} + \mathrm{UV})^\top \mathrm{A}^\top \bar{y} && \left(N^2 + 2NR + 2N \text{ multiply-adds}\right), \\ & \bar{\mathrm{U}} \mathrel{+}= (\mathrm{A}^\top \bar{y})(\mathrm{VB}x)^\top && \left(NR \text{ multiply-adds}\right), \\ & \bar{\mathrm{V}} \mathrel{+}= (\mathrm{U}^\top \mathrm{A}^\top \bar{y})(\mathrm{B}x)^\top && \left(NR \text{ multiply-adds}\right), \\ & \bar{\mathrm{A}} \mathrel{+}= \bar{y} \odot \left((\mathrm{W} + \mathrm{UV})\mathrm{B}x\right) && \left(N \text{ multiply-adds}\right), \\ & \bar{\mathrm{B}} \mathrel{+}= \left((\mathrm{W} + \mathrm{UV})^\top \mathrm{A}^\top \bar{y}\right) \odot x && \left(N \text{ multiply-adds}\right). \end{aligned}$$

Thus, combining the BPT forward pass $\left(N^2 + 2NR + 2N\right)$ and backward pass $\left(N^2 + 4NR + 4N\right)$ gives

$$\mathrm{FLOPs}_{\mathrm{BPT}} = 2N^2 + 6NR + 6N.$$

This is close to the FLOP estimate for LoRA (Schulman and Lab, 2025), which is

$$\mathrm{FLOPs}_{\mathrm{LoRA}} = 2N^2 + 6NR.$$

BPT incurs an additional $6N$ cost due to the diagonal factors. Compared to full training, which costs $3N^2$ per layer, BPT uses

$$\frac{\mathrm{FLOPs}_{\mathrm{BPT}}}{\mathrm{FLOPs}_{\mathrm{Full}}} = \frac{2N^2 + 6NR + 6N}{3N^2} = \frac{2}{3} + \frac{2R}{N} + \frac{2}{N}.$$

When $R \ll N$, this is only slightly larger than $2/3$, meaning that BPT uses roughly two-thirds of the compute of full training. This analysis follows the same setup as (Schulman and Lab, 2025) and considers only FLOPs in linear layers, which account for the majority of FLOPs during training.

D Model Architecture and Parameter Reduction

All models are based on the Qwen2.5 architecture (Qwen et al., 2025): in addition to the standard 0.5B and 1.5B parameter versions, we developed a custom 100M parameter variant by scaling down the same architecture. We also pre-train OLMo-2-1B (OLMo et al., 2024). Although both models are transformer-based auto-regressive language models, they differ slightly in their attention implementation (Grouped Query Attention vs Multi-head Attention). Additionally, Qwen-2.5 ties its input and output embeddings, whereas OLMo-2 has separate input and output embeddings. For OLMo-2, we directly use the official implementation from the authors.

Table 3: Model Architecture Detail and Total Training Tokens

Models	Layers	Heads (Q / KV)	Hidden Size	Intermediate Size	Training Tokens
100M	8	8/ 2	512	2048	10B
0.5B	24	14 / 2	896	4864	10B
1.5B	36	16 / 2	2048	11008	10B and 100B

Table 4: Trainable parameter counts by method and model size

Model	Full Trainable	BPT Trainable	Total Reduction (%)	Non-embedding Reduction (%)
Small	108M	89M	17.8	63.5
Medium	0.5B	0.2B	58.2	80.4
Large	1.5B	0.5B	65.9	77.4

E Hyperparameters and Experimental Setup

For BPT and other baselines, we applied this method exclusively to the linear layers within the transformer blocks. Parameters such as layer normalization, embedding layers, and (lm_head) were kept fully trainable. For the Qwen2.5 architecture, attention matrices, such as the query projection layer `q_head`, have dimensions 518 by 128. Applying BPT to such layers would increase the number of trainable parameters if the low-rank matrices' rank is set to 128, undermining the parameter efficiency goal for these specific matrices. Hence, we leave them unchanged. To ensure that our method is robust across model architectures.

We did not conduct extensive hyperparameter search for learning rate. Following the trend of PEFT papers, where learning rate is higher compared to full fine-tuning. We simply used 10x of the base learning rate which was 2e-4 for all our base models. For ReLoRA with Qwen 1.5B model and LoRA with Qwen 0.5B model, we used a learning rate of 1e-3 after multiple crashes with learning rate 2e-3. Additionally, all the parameters in the model were trained using bf16 including the low-rank matrices with the exception of the diagonal matrices which were kept in fp32 for stability. For a fair comparison, all experiments for a given model size used the same batch size and were trained for the same total number of steps. The optimizer used was AdamW (Loshchilov and Hutter, 2019), and the learning rate followed a warm-up and cosine decay schedule (WSD).

F GPU Memory Footprint

On a 4-GPU NVIDIA GH200 setup, we evaluate the memory consumption of Qwen-2.5-1.5B at approximately 8192 tokens per GPU (16 microbatches $\times$ 512 sequence length). Compared to full training, BPT reduces peak GPU memory by 5.8 GB per GPU with `torch.compile` and by 7.9 GB per GPU with `torch.compile` plus gradient checkpointing, as summarized in Figure 6. These memory savings come with roughly 7% wall-clock overhead under `torch.compile`; because this uses a pure PyTorch implementation without custom kernels, the overhead is likely an upper bound.

Table 5: The hyperparameters used for pre-training Qwen-2.5 Models.

Method	Models	100M	0.5B	1.5B
	Optimizer	AdamW		
	Warmup Steps	2000		
	Decay Steps	10000,8000,8000		
	Max Grad Norm	1.0		
	Max Seq. Len	512		
	Batch Size	320,512,1024		
	LR Schedule	WSD		
	Tokens	10B		
Full	Learning Rate	2e-4		
BPT	Learning Rate	2e-3		
	Rank	128,128,256		
ReLoRA	Learning Rate	2e-3,1e-3		
	Rank	,128,256		
LoRA	Learning Rate	1e-3		
	Rank	128		
HyperAdapt	Learning Rate	2e-3		

Table 6: The hyperparameters used for pre-training OLMo-2-1B.

Method	Models	1B
	Optimizer	AdamW
	Warmup Steps	2000
	Decay Steps	8000
	Max Grad Norm	1.0
	Max Seq. Len	512
	Batch Size	960
	LR Schedule	WSD
	Tokens	10B
Full	Learning Rate	2e-4
BPT	Learning Rate	2e-3
	Rank	256
ReLoRA	Learning Rate	2e-3
	Rank	256

Table 7: The hyperparameters used for pre-training Qwen-2.5 1.5B on 10 billion tokens.

Method	Models	1.5B
	Optimizer	AdamW
	Warmup Steps	2000
	Decay Steps	24,000
	Max Grad Norm	1.0
	Max Seq. Len	512
	Batch Size	1024
	LR Schedule	WSD
	Tokens	100B
Full	Learning Rate	2e-4
BPT	Learning Rate	2e-3
	Rank	256