

# DEEP HIERARCHICAL LEARNING WITH NESTED SUBSPACE NETWORKS

**Paulius Rauba\***  
University of Cambridge

**Mihaela van der Schaar**  
University of Cambridge

## ABSTRACT

Large neural networks are typically trained for a fixed computational budget, creating a rigid trade-off between performance and efficiency that is ill-suited for deployment in resource-constrained or dynamic environments. Existing approaches to this problem present a difficult choice: training a discrete collection of specialist models is computationally prohibitive, while dynamic methods like slimmable networks often lack the flexibility to be applied to large, pre-trained foundation models. In this work, we propose *Nested Subspace Networks (NSNs)*, a novel architectural paradigm that enables a single model to be dynamically and granularly adjusted across a continuous spectrum of compute budgets at inference time. The core of our approach is to re-parameterize linear layers to satisfy a nested subspace property, such that the function computed at a given rank is a strict subspace of the function at any higher rank. We show that this entire hierarchy of models can be optimized jointly via an uncertainty-aware objective that learns to balance the contributions of different ranks based on their intrinsic difficulty. We demonstrate empirically that NSNs can be surgically applied to pre-trained LLMs and unlock a smooth and predictable compute-performance frontier. For example, a single NSN-adapted model can achieve a 50% reduction in inference FLOPs with only a 5 percentage point loss in accuracy. Our findings establish NSNs as a powerful framework for creating the next generation of adaptive foundation models.

## 1 INTRODUCTION

Deploying large neural networks is a central challenge in modern machine learning. For instance, a foundation model (Bommasani, 2021) may be trained on a massive GPU cluster, but we may nevertheless be interested in deploying it in an environment with a strict and dynamic computational budget (Xu et al., 2025). Here, we consider precisely this problem of creating a single network that can flexibly trade off its performance and inference cost. Specifically, consider the general case where a single trained model must serve a variety of deployment scenarios, each with different latency requirements and hardware capabilities (Sze et al., 2017).

Most popular approaches fall into two main categories. On the one hand, conventional approaches operate by creating smaller, static artifacts from a larger pre-trained model (Cheng et al., 2017), using techniques like network pruning (Han et al., 2015b; Blalock et al., 2020) or knowledge distillation (Gou et al., 2021). More recently, parameter-efficient fine-tuning methods like Low-Rank Adaptation (LoRA) (Hu et al., 2022) have gained popularity for adapting large models, but these also produce a static, low-rank adaptation for a fixed budget. In theory, this approach yields highly optimized models for a specific computational target. In practice, however, this strategy suffers from its static nature; creating a model for a new budget requires repeating the entire, often costly, compression pipeline (Zhu & Gupta, 2017), and it fails to provide the granular, on-the-fly adaptability needed for dynamic environments.

On the other hand, recent methods using dynamic neural networks (Han et al., 2021) operate by designing architectures that can be adjusted at inference time, such as slimmable networks that can drop channels (Yu et al., 2018; Li et al., 2021) or layers (Wu et al., 2018). In theory, these approaches more readily take advantage of a single set of weights to serve multiple budgets. In practice, however,

---

\*Corresponding author: pr501@cam.ac.uk

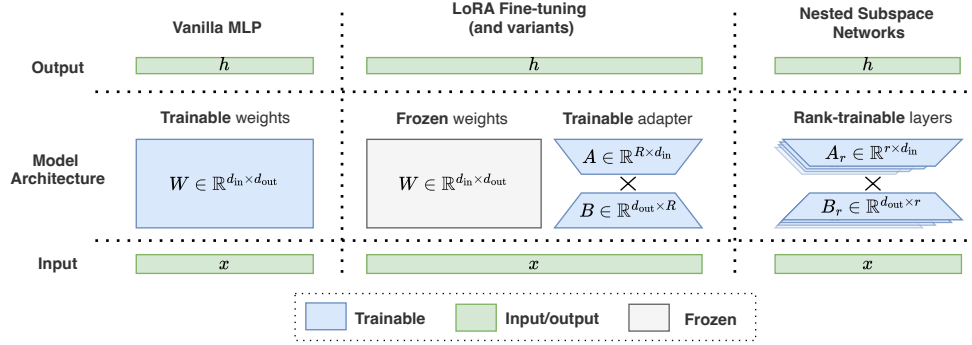


Figure 1: **Illustration of Nested Subspace Networks.** NSNs convert linear layers into rank-trainable layers which enable dynamic control over the computational cost (FLOPs) of a forward pass. **Left:** Standard MLP layers that are composed of trainable weights. **Middle:** LoRA fine-tuning which have frozen weights and trainable adapters. **Right:** Nested Subspace networks *replace linear layers with rank-trainable layers* and learn to interpolate across many ranks at test time. This allows for the construction of a compute-performance Pareto frontier at inference time.

this strategy often comes at the price of much more challenging, specialized training schemes that are typically applied from scratch (Cai et al., 2019). This makes them difficult to apply to the vast ecosystem of existing, pre-trained foundation models, which represent the vast majority of trained and used models today. Further, many of these techniques—with some notable exceptions (Yu & Huang, 2019)—offer only a coarse, discrete set of operating points rather than a smooth, continuous trade-off (Teerapittayanon et al., 2016; Yu et al., 2018).

**Three Desiderata** Can we develop a better approach? Building on the discussion above, we contend that a good solution to the dynamic inference problem should satisfy the following three desiderata.

- **D1: Instant Adaptability.** Instantly trade-off compute and performance at test-time without any additional overhead or expensive fine-tuning procedures in a single neural architecture.
- **D2: Post-Hoc Applicability.** Have the architectural generality to be applied to any pre-trained foundation model and be widely applicable for many classes of models.
- **D3: Granularity.** It should provide a smooth, continuous spectrum of operating points along the compute-performance Pareto frontier, not just a few discrete, pre-determined choices.

In this work, we present an effective method that satisfies these criteria and introduces a new paradigm of flexible model deployment. Our contributions are three-fold.



**Contributions.** **First**, we introduce Nested Subspace Networks (NSNs), a novel architecture that represents a continuous hierarchy of models within a single set of weights, and we propose a practical uncertainty-aware training objective that makes this hierarchy learnable (Sec. 2). **Second**, we provide theoretical guarantees for granular budget control, showing that our method induces a smooth and predictable performance-compute frontier, even for budgets not explicitly seen during training (Sec. 3). **Third**, we demonstrate the broad utility and effectiveness of NSNs through comprehensive experiments (Sec. 4), including the surgical adaptation of large pre-trained language models, and show that a single adaptive network can match the performance of multiple specialist models.

## 2 NESTED SUBSPACE NETWORKS

**Preliminaries.** Consider a standard feed-forward neural network. A standard linear layer computes the affine transformation  $f(\mathbf{x}) = W\mathbf{x} + \mathbf{b}$ , where  $\mathbf{x} \in \mathbb{R}^{d_{in}}$  is the input vector, the weight matrix is  $W \in \mathbb{R}^{d_{out} \times d_{in}}$ , and the bias  $\mathbf{b} \in \mathbb{R}^{d_{out}}$ . The number of parameters in  $W$  scales with the product of the input and output dimensions which becomes a large computational and memory bottleneck. This motivates the need for efficient parametrizations.

**Low-rank factorization** has become a popular approach to mitigating the quadratic cost (Hu et al., 2022), where the full-rank matrix  $W$  is approximated with two smaller matrices  $W = BA$ , where  $A \in \mathbb{R}^{R \times d_{\text{in}}}$  and  $B \in \mathbb{R}^{d_{\text{out}} \times R}$ . Here,  $R \ll \min(d_{\text{in}}, d_{\text{out}})$  is a maximum rank. The transformation becomes  $f(\mathbf{x}) = (BA)\mathbf{x} + \mathbf{b} = B(A\mathbf{x}) + \mathbf{b}$ .

## 2.1 THE NESTED SUBSPACE ARCHITECTURE

NSNs are a class of neural network architectures designed for parameter efficiency and dynamic, post-training adjustment of model capacity. The core principle is to re-parameterize a linear layer with a sequence of low-rank approximations  $\{W_r\}_{r=1}^R$  that form a *nested hierarchy*, such that the image of each approximation is a subspace of the next. The architecture is built on the principle of low-rank factorization which we extend to overcome its static limitations and make it applicable to a wide variety of network architectures.

**Reducing FLOPs.** The low-rank factorization reduces the model’s active parameter count and, consequently, its required floating-point operations (FLOPs). This yields a reduction when  $r$  is below the break-even point:  $2r(d_{\text{in}} + d_{\text{out}}) < 2d_{\text{in}}d_{\text{out}}$ , which defines the break-even rank as  $\frac{d_{\text{in}}d_{\text{out}}}{d_{\text{in}} + d_{\text{out}}}$ .

**Definition 1 (Nested Subspace Network).** A Nested Subspace Network (NSN) is a neural network architecture that incorporates one or more **NSN layers**. An NSN layer is a linear transformation parameterized by a pair of factor matrices,  $A \in \mathbb{R}^{R \times d_{\text{in}}}$  and  $B \in \mathbb{R}^{d_{\text{out}} \times R}$ , where  $R$  is a fixed maximum rank. For a rank  $r \in \{1, \dots, R\}$ , the effective weight matrix  $W_r \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$  is constructed from the submatrices  $A_r$  (the first  $r$  rows of  $A$ ) and  $B_r$  (the first  $r$  columns of  $B$ ). This is expressed as a sum of the rank-1 outer products, i.e.  $W_r := B_r A_r = \sum_{i=1}^r \mathbf{b}_i \mathbf{a}_i$  where  $\mathbf{a}_i \in \mathbb{R}^{1 \times d_{\text{in}}}$  is the  $i$ -th row of  $A$  and  $\mathbf{b}_i \in \mathbb{R}^{d_{\text{out}} \times 1}$  is the  $i$ -th column of  $B$ .

NSNs are a flexible class of models that can operate on model architecture as long as it comprises linear layers. Therefore, it is applicable to models of different sizes, architectures, purposes, with varying inductive biases, etc. A central feature of NSNs is that it naturally gives rise to a fundamental property that we exploit in this work: the *nested subspace property*.

**Definition 2 (Nested Subspace Property).** The family of weight matrices  $\{W_r\}_{r=1}^R$  generated by an NSN layer satisfies the **nested subspace property** if the image of the rank- $r$  transformation is a subspace of the image of the rank- $(r + 1)$  transformation for all  $1 \leq r < R$ :

$$\text{Im}(W_r) \subseteq \text{Im}(W_{r+1}) \quad \forall r \in \{1, \dots, R-1\}$$

This implies the existence of a filtration of vector spaces:  $\text{Im}(W_1) \subseteq \text{Im}(W_2) \subseteq \dots \subseteq \text{Im}(W_R)$ .

What does this property mean in practice? NSN layers parametrize *an entire hierarchy of models*. In this hierarchy, due to the nested subspace property, the function class realized by a rank- $r$  model is a strict subset of the function class of a rank- $(r + 1)$  model. Therefore, with the right training scheme, we can choose “which network” we want to employ by deciding on the rank.

However, this approach raises an immediate issue: *how can we train a model that learns a hierarchy of nested sub-models?* A naive approach would be to parametrize the model via low-rank factorization, select a highest rank, train it on the highest rank, and at test time truncate it to the desired rank  $r$ . While this technically satisfies the nested subspace property, there is no inductive bias to make the models operate along the compute-efficiency Pareto frontier (see Fig 2, implementation details can be found in Appendix A.1.) An alternative approach could be to train *simultaneously* at different ranks and sum cross entropies within each rank. However, such a

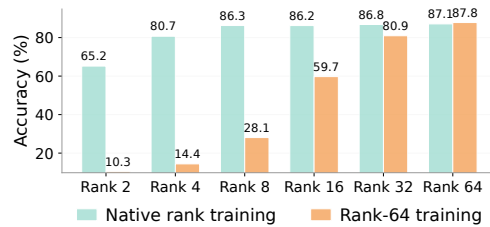


Figure 2: **Comparison of native-rank training and rank truncation for an MLP on CIFAR-10.** The plot compares the accuracy of individually training a model for each specific rank (*Native rank training*) versus training a single model at a high rank (64) and truncating it to lower ranks at test time (*Rank-64 training*). The significant performance gap demonstrates that naively truncating a high-rank model results in poor performance.

naive approach suffers from at least three main issues: (i) it does not take into account the intrinsic difficulty of learning a lower rank model (harder) *versus* a higher rank model (easier); (ii) It results in training instability resulting from large losses in low-rank models; (iii) It is computationally prohibitive to train at all possible ranks. In the next section, we propose an uncertainty-aware training procedure that resolves these challenges for training NSNs.



**Takeaway.** NSNs create a hierarchy of models within a single network using the *nested subspace property*. The key challenge is training one set of weights to be optimal across this entire hierarchy simultaneously.

## 2.2 TRAINING WITH MULTI-RANK UNCERTAINTY

Our central goal is to learn a single parametrization that simultaneously yields optimal performance for all sub-models. We posit that the failure of naive approaches stems from the differences in intrinsic difficulty of learning different ranks (Sec. 2.1). Therefore, we treat the optimization of each sub-model at different ranks as a multi-task learning problem with varying difficulty levels. One way to reframe the “difficulty” of a problem is by quantifying the *aleatoric uncertainty* of each task and weighting each task in proportion to that uncertainty (Kendall et al., 2018). We model the aleatoric uncertainty by introducing learnable variance parameters  $\sigma_k^2$  for each rank  $k$ . This variance is assumed to be *heteroskedastic across ranks* (i.e.,  $\sigma_k^2 \neq \sigma_j^2$  for  $k \neq j$ ) but *homoskedastic within a rank* (i.e., constant for all inputs).

**Modeling assumption.** We consider the classification case (the regression case is analogous). Following Kendall et al. (2018), we use the standard uncertainty-weighted *surrogate* objective in which each rank’s cross-entropy is weighted by a learnable scale and regularized by a corresponding log-term. Concretely, for rank  $k$  we use the per-rank contribution:

$$\frac{1}{2\sigma_k^2} \mathcal{L}_{\text{CE}}(k) + \log \sigma_k, \quad (1)$$

which serves purely as a weighting-and-regularization surrogate for balancing tasks in classification (i.e., it is not interpreted as a probabilistic likelihood over  $\mathcal{L}_{\text{CE}}$ ).

**Formulating the uncertainty-weighted training objective.** During training, we sample an *anchor rank*  $\tilde{R} \leq R$ —the maximum rank used at training time—and a *variant rank*  $r < \tilde{R}$ . Assuming independent uncertainty parameters across ranks, the total objective for a training step is the sum of the two surrogate terms:

$$\left( \frac{1}{2\sigma_{\tilde{R}}^2} \mathcal{L}_{\text{CE}}(\tilde{R}) + \log \sigma_{\tilde{R}} \right) + \left( \frac{1}{2\sigma_r^2} \mathcal{L}_{\text{CE}}(r) + \log \sigma_r \right). \quad (2)$$

We reparameterize the variance by learning its logarithm  $s_k = \log(\sigma_k^2)$  and drop the constant factor  $\frac{1}{2}$  (Kendall et al., 2018). This results in our final training objective, a function of the shared weights  $A$  and  $B$  (which define the model weights) and the learnable log-variances:

$$\mathcal{L}_{\text{total}}(A, B, s_{\tilde{R}}, s_r) = (\exp(-s_{\tilde{R}}) \mathcal{L}_{\text{CE}}(\tilde{R}) + s_{\tilde{R}}) + (\exp(-s_r) \mathcal{L}_{\text{CE}}(r) + s_r). \quad (3)$$

**Insights on the formulated objective.** The uncertainty-weighted surrogate implicitly performs gradient balancing across ranks, since the effective contribution of each term scales with  $\exp(-s_k)$ . This connects our objective to established approaches for multi-task optimization that seek Pareto-stationary solutions by equilibrating task gradients (Sener & Koltun, 2018), as well as to adaptive weighting schemes such as GradNorm (Chen et al., 2018) and Dynamic Weight Averaging (Liu et al., 2019). This promotes the well-behaved performance frontier we analyze in Sec. 4.3.

In practice, the inclusion of an anchor rank significantly stabilizes training and helps to learn a better final higher-rank model. Furthermore, we find that introducing a curriculum-learning-based sampling strategy for the variant ranks substantially improves downstream results relative to uniform sampling.

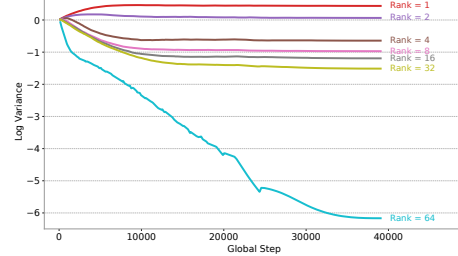


**Takeaway.** We formalize the joint optimization of different ranks by learning rank-specific homoskedastic variance via a standard uncertainty-weighted surrogate.

### 2.3 INTERPRETING LOG VARIANCES AS A PROXY FOR RANK EXPRESSIVENESS

*What has this formulation achieved?* Now, the learned log variances *modulate* the influence of each rank’s loss contribution to the objective. High log-variance promotes gradient attenuation by scaling the cross-entropy loss from high-variance ranks. Low log-variance amplifies the penalty loss for cross-entropy losses by increasing its magnitude. In fact, we can directly quantify the gradient contributions based on the loss as:

$$\nabla_w \mathcal{L}_{\text{total}} = \underbrace{\exp(-s_{\tilde{R}}) \nabla_w \mathcal{L}_{CE}(\tilde{R})}_{\text{Anchor Contribution}} + \underbrace{\exp(-s_r) \nabla_w \mathcal{L}_{CE}(r)}_{\text{Variant Contribution}}$$



**Figure 3: Learned log-variances during training with the multi-rank uncertainty objective on CIFAR-10 dataset.** We train a single model with an anchor and variant ranks and find that higher ranks have lower task-dependent uncertainty during training.

This mechanism allows the learned log-variances to serve as an emergent proxy for the *effective expressiveness* of each rank-specific model. It follows that higher ranks, which possess greater representational capacity, should learn lower corresponding variances. As we demonstrate empirically, this is precisely the behavior we observe in our experiments (Fig. 3).

## 3 PERFORMANCE INTERPOLATION BETWEEN RANKS

A central claim of our work is that NSNs provide granular control over the compute-performance trade-off. This implies not only strong performance at a discrete set of trained ranks but also reliable behavior at *interpolated* ranks that were not explicitly part of the optimization objective. In the absence of theoretical guarantees, one might expect performance to be unpredictable or even collapse between these well-trained points. In this section, we provide the formal underpinnings for the smooth and well-behaved nature of the performance-compute frontier induced by NSNs. We begin by introducing a mild assumption on the structure of the learned weights, from which we derive a formal bound on the interpolation error.

To formalize the notion of a smooth frontier, we must first characterize the structure that our multi-rank uncertainty training (Section 2.2) imposes on the parameterization. We posit that the optimization encourages a natural ordering of basis vectors by importance.

**Assumption 1** (Rank-1 Component Energy Decay). *The training procedure yields a parameterization where the norms of the rank-1 component vectors are monotonically non-increasing with their index. For any indices  $i, j$  such that  $1 \leq i < j \leq \tilde{R}$ :*

$$\|\mathbf{a}_j\| \leq \|\mathbf{a}_i\| \quad \text{and} \quad \|\mathbf{b}_j\| \leq \|\mathbf{b}_i\|$$

Our training objective motivates this assumption, since it naturally encourages the model to allocate the most salient information to the lowest-indexed basis vectors, as they must be utilized by all nested sub-models.

By extending this result from the model’s output to its expected loss, and assuming a standard regularity condition on the loss function, we can establish a bound on the difference in expected performance between any two ranks.

**Proposition 1** (Bound on Interpolation Error). *Let the task loss function  $\mathcal{L}(f(\mathbf{x}; r), y)$  be  $L_{\mathcal{L}}$ -Lipschitz continuous with respect to its first argument. Let  $E(r) = \mathbb{E}_{(\mathbf{x}, y)}[\mathcal{L}(f(\mathbf{x}; r), y)]$  be the expected error at rank  $r$ . For any ranks  $r_1 < r_{\text{int}} < R$ , the difference in expected error is bounded*

by:

$$|E(r_{int}) - E(r_1)| \leq C \sum_{i=r_1+1}^{r_{int}} \|\mathbf{b}_i\| \|\mathbf{a}_i\|$$

where  $C = L_{\mathcal{L}} \cdot \mathbb{E}[\|\mathbf{x}\|]$  is a task-dependent constant.

Proof in Appendix B.1. This result provides a formal guarantee that the variation in model performance is controlled by the cumulative energy of the intermediate basis vectors. This is important because it justifies the use of NSNs for reliable control across a continuous spectrum of computational budgets. We empirically evaluate this claim in Sec. 4.3.



**Takeaway.** Our training objective induces a smooth and predictable trade-off between computational budget and model performance even on untrained ranks

## 4 EXPERIMENTAL EVALUATION

### 4.1 EVALUATING NSN PERFORMANCE

The primary goal of this experiment is to determine whether a single NSN, trained with our multi-rank uncertainty objective, can match the performance of multiple, specialized models that are individually trained for fixed computational budgets.

**Setup.** We conduct our evaluation on an image classification task using a standard multi-layer perceptron (MLP) architecture. For each computational budget, corresponding to rank  $r \in \{1, 2, \dots, 64\}$ , we train a separate, standard MLP from scratch whose layers are sized to match the FLOPs of an NSN layer at that specific rank on CIFAR-10 (*Base MLP*). Inputs are ImageNet last layer embeddings. This collection of individually optimized models represents an empirical Pareto frontier for the compute-performance trade-off. We then train a single NSN model using the defined objective (*Low Rank Layer*).

**Results.** The results are presented in Fig. 4.

We show that our single, dynamically adjustable NSN achieves performance that is highly competitive with the series of individually trained baseline networks across all tested computational budgets. Therefore, a single NSN can be deployed and dynamically configured at test time to occupy various points on the compute-performance curve.

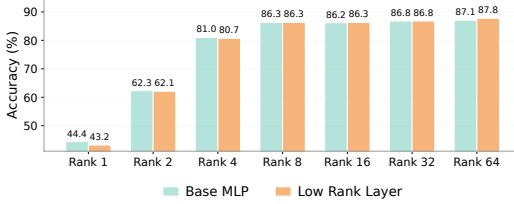


Figure 4: **Test accuracy comparison between a single NSN evaluated at different ranks (orange) and a series of individually trained MLPs with equivalent FLOPs (blue).** The single NSN effectively matches the performance of multiple specialized models, validating its ability to learn a hierarchy of optimal sub-networks.

### 4.2 ABLATION STUDIES

To better understand the mechanisms behind our proposed training strategy, we conduct a series of ablation studies. Our goal is to isolate the contribution of the core components of our objective function and to validate our design choices against plausible alternatives. We compare several training variations, evaluating their impact not only on the highest-rank model but, more importantly, on the average performance of the resulting lower-rank and interpolated sub-models.

**Setup.** We use the same dataset and setup as in Sec. 4.1. We assess each method on three metrics: (i) the final test accuracy of the highest-rank (anchor) model; (ii) the average accuracy across all in-distribution (ID) sub-ranks evaluated during training; and (iii) the average accuracy across out-of-distribution (OOD) interpolated ranks not explicitly seen during training. The results are summarized in Table 1.

**Results.** Our analysis in Table 1 shows that the key to effective sub-model performance is the joint optimization of an anchor and a variant rank ("Two CEs"). This simple objective acts as an implicit



Table 1: Ablation study of different training objectives. Our core Two CEs formulation (highlighted) dramatically improves the performance of lower-rank (ID) and interpolated (OOD) sub-models.  $\tilde{R}$  denotes the anchor rank and  $r$  the variant rank. Performance is reported as mean  $\pm$  std. dev.

Method	Key Formulation	Highest Test Acc	Avg. ID Acc	Avg. OOD Acc
<i>Baselines</i>				
CE Only (Anchor)	$\mathcal{L}_{\text{CE}}(\tilde{R})$	$0.87 \pm 0.00$	$0.48 \pm 0.00$	$0.57 \pm 0.00$
One CE + Hard Ortho.	$+\ AA^T - I\ _F^2$	$0.87 \pm 0.00$	$0.42 \pm 0.00$	$0.50 \pm 0.00$
<i>Variations on Joint Training</i>				
Two CEs	$\mathcal{L}_{\text{CE}}(\tilde{R}) + \mathcal{L}_{\text{CE}}(r)$	<b><math>0.88 \pm 0.00</math></b>	<b><math>0.79 \pm 0.00</math></b>	<b><math>0.81 \pm 0.00</math></b>
+ Logits Regularization	$+\ \text{logits}(\tilde{R}) - \text{logits}(r)\ _2^2$	$0.87 \pm 0.00$	$0.64 \pm 0.00$	$0.64 \pm 0.00$
+ Residual Orthogonality	$+\ A_r A_{\text{res}}^T\ _F^2$	<b><math>0.88 \pm 0.00</math></b>	$0.78 \pm 0.00$	$0.80 \pm 0.00$
+ Hidden Regularization	$+\ \mathbf{h}_{\tilde{R}} - \mathbf{h}_r\ _2^2$	<b><math>0.88 \pm 0.00</math></b>	<b><math>0.79 \pm 0.00</math></b>	$0.79 \pm 0.00$

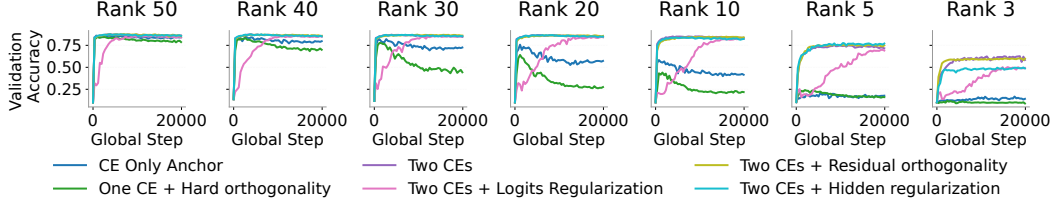


Figure 5: **Training Dynamics at Interpolated Ranks.** Validation accuracy during training for various objective functions, evaluated at ranks not explicitly optimized for. Our proposed method maintains stable learning across all ranks, while simpler baselines exhibit instability and performance collapse.

regularizer. In contrast, we found that adding explicit regularization terms on top of our proposed objective was either redundant or detrimental. This supports our claim that the joint optimization of multiple ranks is a sufficient mechanism for learning NSNs.

#### 4.3 EVALUATING INTERPOLATED RANKS

We now empirically validate the theoretical guarantees for smooth interpolation presented in Sec. 3. We ask whether our method yields robust performance even at ranks that were not explicitly part of the optimization process, thereby satisfying our desideratum for granular budget control (Sec 1).

**Setup.** To investigate this, we analyze the performance of models trained with the various objective functions from our ablation study. We train a single model using each objective, which involves a sparse sampling of ranks. We then evaluate the resulting model not only on the anchor rank but also across a wide spectrum of intermediate, interpolated ranks to assess the stability and smoothness of the learned performance curve.

**Results.** Our findings show that the choice of training objective is important for achieving stable interpolation. As illustrated in Figure 5, baseline objectives that do not properly balance the learning dynamics across ranks often result in unstable or collapsing performance at these intermediate points. For example, training with a single cross-entropy objective leads to poor generalization at lower ranks. In contrast, our proposed objective, which jointly trains an anchor and variant rank, produces stable and monotonically improving accuracy curves across the entire hierarchy of ranks throughout training.

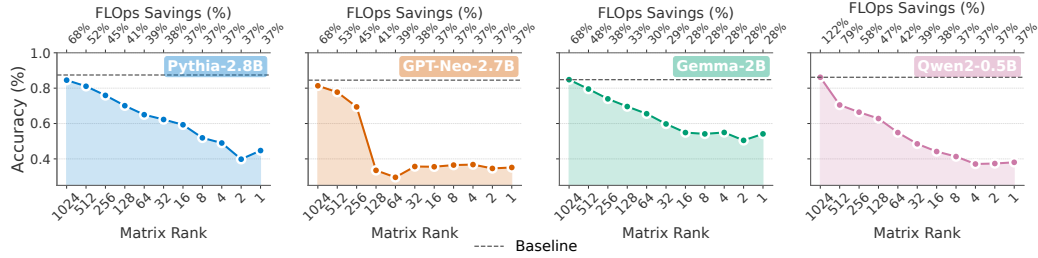


Figure 7: **Accuracy vs rank trade-off for pre-trained LLMs with surgically adapted NSN layers.** We can obtain large reduction in computational cost with minimal decreases in performance.

#### 4.4 APPLICATION TO PRE-TRAINED LLMs

We now evaluate the post-hoc applicability of NSNs to large, pre-trained language models (LLMs).

**Adapting Pre-trained Layers.** Our procedure for adapting a pre-trained LLM consists of surgically replacing the standard linear layers within its MLP blocks with NSN layers. A naive approach might be to randomly initialize the NSN factor matrices  $A$  and  $B$ . In practice, however, this method discards the information encoded in the pre-trained weights. To preserve such information in the pre-trained weights, we initialize the NSN factor matrices using Singular Value Decomposition (Appendix B.2).

**Setup.** We apply this adaptation procedure to four publicly available LLMs: *Pythia-2.8B*, *GPT-Neo-2.7B*, *Gemma-2B*, and *Qwen2-0.5B*. After replacing and initializing the linear layers in their MLP blocks as described above, we fine-tune each model on a downstream task. For our primary analysis with Pythia-2.8B, we use a Natural Language Inference (NLI) benchmark which requires a three-way classification to determine if a premise entails, contradicts, or is unrelated to a given hypothesis. The fine-tuning for all models uses the uncertainty-aware objective described in Section 2.2.

**Results.** Our results demonstrate that NSNs unlock a smooth and predictable compute-performance frontier for large, pre-trained models. For instance, Pythia-2.8B exhibits a monotonic degradation in accuracy as the rank—and therefore the operational FLOPs—is reduced (Fig. 11). This granular control allows for substantial efficiency gains with a modest performance trade-off; for instance, a 50% reduction in computational cost is achieved with only a 5 percentage point drop in accuracy. We find this behavior is consistent across all four tested language models, where performance drops smoothly as the matrix rank is decreased. This establishes NSNs as an effective method for post-hoc adaptation of foundation models to dynamic inference scenarios.

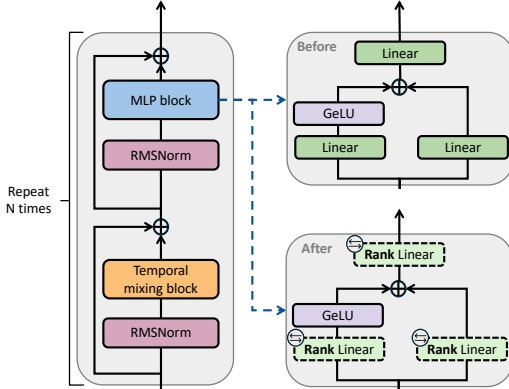


Figure 6: **Example Surgical Changes to linear layers only on Gemma-2B.** The architecture of Gemma-2B. All MLP blocks contain three linear layers with a GeLU activation function. We surgically replace all linear layers with *rank-adaptive* linear layers, initialized  $W \approx BA$  via SVD-decomposition



**Takeaway.** NSNs can be surgically applied to large, pre-trained foundation models, which allows for smooth and predictable compute-performance trade-offs.

## 5 RELATED WORK

On the one hand, static compression methods aim to create smaller, more efficient, but ultimately fixed models from a larger pre-trained one. Techniques like **network pruning** (Han et al., 2015b; Blalock et al., 2020) and **knowledge distillation** (Gou et al., 2021) excel at this, producing highly



Table 3: **A comparative analysis against our three desiderata for efficient, adaptable architectures.** Recall the three desiderata where we seek a solution that: **D1**: learns a single, unified *Trade-off Parametrization*<sup>(1)</sup> that allows for instant *Test-time adaptability*<sup>(1)</sup>; **D2**: is broadly applicable through *Post-Training Re-parameterization*<sup>(2)</sup> and exhibits *Architectural Agnosticism*<sup>(2)</sup> to modify existing pre-trained models; and **D3**: provides granular control by generating a *Smooth trade-off Frontier*<sup>(3)</sup> across a continuous spectrum of computational budgets. Our proposed method, Nested Subspace Networks (NSNs), is the first to satisfy all five criteria.

	Method	Example	Trade-off Parametrization <sup>(1)</sup>	Test-time Adaptability <sup>(1)</sup>	Post-Training Re-parameterization <sup>(2)</sup>	Architectural Agnosticism <sup>(2)</sup>	Smooth Trade-off Frontier <sup>(3)</sup>
LoRA	Standard LoRA	(Hu et al., 2022)	✗	✗	✓	✓	✗
	DyLoRA	(Valipour et al., 2022)	✗	✓	✓	✓	✓
	LoRA-Pruning	(Chen et al., 2023)	✗	✗	✓	✓	✗
Other	Universal Slimmable	(Yu & Huang, 2019)	✓	✓	✗	✗	✓
	Once-for-All (OFA)	(Cai et al., 2019)	✓	✓	✗	✗	✓
	Iterative Magnitude	(Han et al., 2015a)	✗	✗	✓	✓	✗
	Movement Pruning	(Sanh et al., 2020)	✗	✗	✗	✓	✗
	Response-Based KD	(Hinton et al., 2015)	✗	✗	✓	✓	✗
	Self-Distill. (Early Exit)	(Teerapittayanon et al., 2016)	✗	✓	✗	✓	✗
	<b>Nested Subspace Networks</b>	<b>Ours</b>	✓	✓	✓	✓	✓

optimized artifacts for a specific computational target. However, this approach is fundamentally static; adapting the model to a new computational budget requires repeating the entire, often costly, compression pipeline, failing to provide the on-the-fly adaptability needed for dynamic environments.

On the other hand, **dynamic neural networks** (Han et al., 2021) are designed with inference-time adaptability in mind. **Slimmable networks**, for instance, allow for channels to be dropped dynamically to create sub-networks of varying widths (Yu et al., 2018). While these methods offer the desired adaptability, they typically require specialized and complex training schemes that are applied from scratch (Cai et al., 2019). This makes them difficult to apply to the vast ecosystem of existing, pre-trained foundation models. Furthermore, many such techniques offer only a coarse, discrete set of operating points rather than a smooth, continuous trade-off (Teerapittayanon et al., 2016).

More recently, parameter-efficient fine-tuning (PEFT) methods like **Low-Rank Adaptation (LoRA)** (Hu et al., 2022) have become a popular way to efficiently adapt large models. While LoRA also employs low-rank factorization, its goal is to learn a *single, static update* for a *fixed* rank  $r$ . It is not designed to be dynamically adjusted at inference time; changing the computational budget would require training a new LoRA adapter with a different rank. In contrast, NSNs leverage a nested low-rank parameterization to enable a single model to be granularly and dynamically adjusted across an entire spectrum of ranks at test time.

## 6 DISCUSSION

The dominant paradigm in deploying large models involves creating static artifacts, each trained for a fixed computational budget. This approach is ill-suited for dynamic environments where resource constraints can change on-the-fly. Contrary to the view that this requires training a discrete collection of specialist models—a computationally prohibitive approach—we make a strong case that a single, well-trained dynamic network can effectively and efficiently navigate this trade-off.

We introduced Nested Subspace Networks (NSNs), a novel architectural paradigm that represents a continuous hierarchy of models within a single set of weights. We propose a structural design based on the nested subspace property that has a practical, uncertainty-aware training objective. We show how an entire family of models can be optimized jointly. We further demonstrated that NSNs can be surgically applied post-hoc to large, pre-trained foundation models, unlocking a smooth and predictable compute-performance frontier without requiring training from scratch. This paper presents the first-ever approach to dynamically convert any pre-trained foundation model to a compute-adjustable model with minimal fine-tuning.

*Why and how do NSNs work so well?* We probe this question in our insights experiments (Sec. A.1). Our analysis reveals that NSNs, regularized by their low-rank structure, converge to different local minima in the loss landscape compared to standard fine-tuning. This means that NSNs find distinct yet highly effective solutions in the loss landscape that allow the family of nested sub-models to work well (Sec. A.4). We empirically verified the foundational nested subspace property, confirming that

the vector spaces of lower-rank models are indeed contained within those of higher-rank models, as intended by the architectural design (Sec. A.3). Furthermore, we attempt to understand what happens to the network structure as we change the rank of the network. We find that the low-rank constraint acts as a bottleneck that encourages layers to learn redundant, globally useful functions (Sec. A.2). As capacity increases with higher ranks, layers diverge, adopting more specialized roles. Our findings and insights establish NSNs as a powerful framework for creating the next generation of adaptive foundation models.

## REFERENCES

- Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Gutttag. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.
- Rishi Bommasani. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.
- Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.
- Tianyi Chen, Tianyu Ding, Badal Yadav, Ilya Zharkov, and Luming Liang. Lorashear: Efficient large language model structured pruning and knowledge recovery. *arXiv preprint arXiv:2310.18356*, 2023.
- Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International conference on machine learning*, pp. 794–803. PMLR, 2018.
- Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.
- Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International journal of computer vision*, 129(6):1789–1819, 2021.
- Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.
- Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015b.
- Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(11): 7436–7456, 2021.
- Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.
- Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7482–7491, 2018.
- Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Dynamic slimmable network. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pp. 8607–8617, 2021.
- Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1871–1880, 2019.
- Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in neural information processing systems*, 33:20378–20389, 2020.

- Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31, 2018.
- Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.
- Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, pp. 2464–2469. IEEE, 2016.
- Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. *arXiv preprint arXiv:2210.07558*, 2022.
- Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8817–8826, 2018.
- Mengwei Xu, Dongqi Cai, Wangsong Yin, Shangguang Wang, Xin Jin, and Xuanzhe Liu. Resource-efficient algorithms and systems of foundation models: A survey. *ACM Computing Surveys*, 57(5): 1–39, 2025.
- Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1803–1811, 2019.
- Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.
- Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

## A ADDITIONAL EXPERIMENTS

### A.1 DETAILS ON NATIVE-RANK TRAINING VS RANK-TRUNCATION

The experiment presented in Figure 2 aims to contrast two approaches for obtaining models at various computational budgets: (i) native-rank training and (ii) rank truncation. All experiments were conducted on the CIFAR-10 dataset using ImageNet embeddings as input to a Multi-Layer Perceptron (MLP).

**Native Rank.** For the "Native rank training" baseline, we trained a series of independent specialist models. Each model corresponds to a specific rank  $r \in \{1, 2, \dots, 64\}$ . The linear layers of each MLP were parameterized using a low-rank factorization  $W = BA$ , where the inner dimension was fixed to the target rank  $r$ . Every model was trained from scratch for 30 epochs using the same hyperparameters to represent the empirical Pareto frontier of performance for a given rank.

**Rank Truncation.** For the "Rank-64 training" comparison, we trained a single model with a maximum rank of  $R = 64$ . At test time, to evaluate performance at a lower rank  $r < R$ , we simply truncated the factor matrices  $A \in \mathbb{R}^{R \times d_{in}}$  and  $B \in \mathbb{R}^{d_{out} \times R}$ . Specifically, the truncated weight matrix  $W_r$  was constructed using only the first  $r$  rows of  $A$  and the first  $r$  columns of  $B$ . This ensures the nested subspace property is structurally satisfied, but as Figure 2 shows, this naive approach fails to train the shared parameters to be effective across the hierarchy of ranks.

### A.2 INTER-LAYER WEIGHT SIMILARITY VS. MATRIX RANK

**Objective.** This experiment investigates the relationship between the representational capacity of layers and their functional roles within the network. The core question is whether increasing layer capacity (i.e., matrix rank) encourages layers to learn more specialized, distinct functions or more similar, redundant ones. The guiding hypothesis is that a low-rank constraint acts as an informational bottleneck, forcing layers to learn redundant, globally useful functions, while increasing the rank enables and encourages functional specialization.

**Methodology.** For a trained Nested Subspace Network, we analyzed the weight matrices of the MLP layers at various ranks. For each rank  $r$  from 1 to 1024, we reconstructed the effective weight matrix  $W_r$  for each layer. We then computed the pairwise cosine similarity between the weight matrices of all layers in the network. The average of these pairwise similarities was then plotted against the matrix rank to observe the overall trend.

**Results and Interpretation.** The results, shown in Figure 8, confirm the hypothesis. At very low ranks, the average inter-layer similarity is high, indicating that the network's layers learn functionally similar and redundant representations. As the matrix rank and thus the layer capacity increase, the average cosine similarity between layers steadily decreases, approaching zero at the highest ranks. This suggests that with greater representational freedom, layers diverge to assume more specialized roles within the network. The low-rank constraint effectively regularizes the network, forcing layers to cooperate on learning general features, while higher ranks allow for a more distributed and specialized division of labor.

### A.3 EMPIRICAL VERIFICATION OF THE NESTED SUBSPACE PROPERTY

**Objective.** The central design of Nested Subspace Networks relies on the nested subspace property, where the function computed at a given rank is a strict subspace of the function at any higher rank. This experiment was designed to empirically verify if this theoretical property holds in practice after training. The key question is: does the vector space spanned by a lower-rank weight matrix truly lie inside the vector space of a higher-rank matrix from the same trained layer?

**Methodology.** To quantify the degree of subspace containment, we used a three-step procedure for each trained NSN layer:

1. **Reconstruct Weights:** For a pair of ranks,  $r_{\text{small}}$  and  $r_{\text{large}}$ , we reconstructed their effective weight matrices,  $W_{r_{\text{small}}}$  and  $W_{r_{\text{large}}}$ .

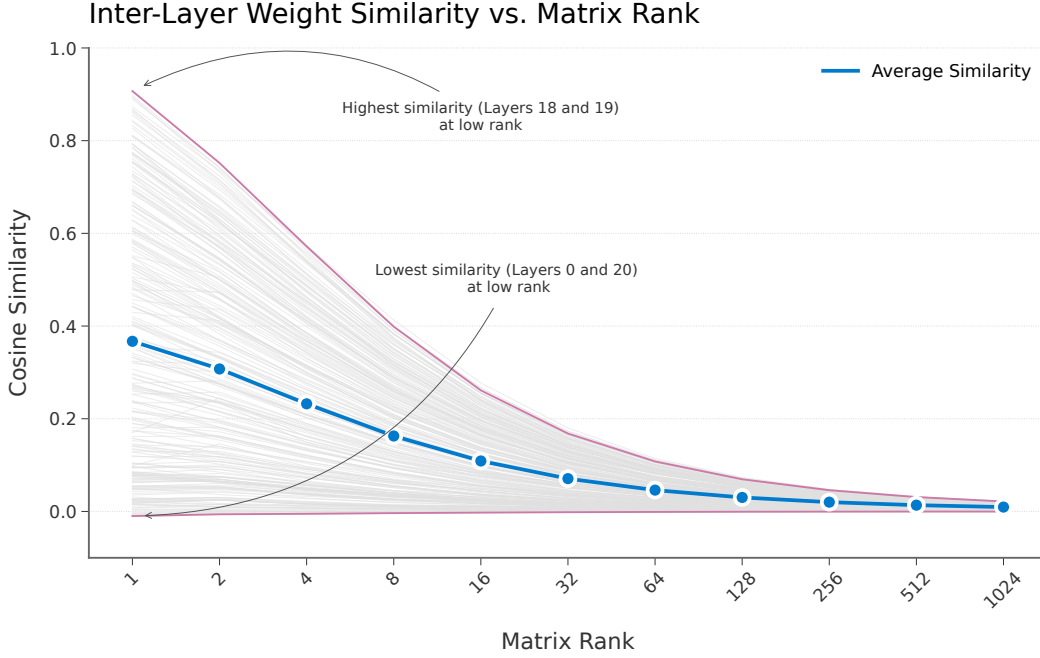


Figure 8: Inter-layer weight similarity as a function of matrix rank. As layer capacity (rank) increases, the average cosine similarity between layers decreases. This suggests that layers transition from learning redundant, globally useful functions at low ranks to more specialized roles at high ranks.

2. **Find Orthonormal Bases:** We performed Singular Value Decomposition (SVD) on each weight matrix ( $W_r = U_r \Sigma_r V_r^T$ ) to find an orthonormal basis for its column space. The first  $r$  columns of the resulting  $U_r$  matrix form this basis.
3. **Calculate Containment Score:** We computed a containment score to measure the extent to which the smaller subspace is contained in the larger one. The score is defined as the normalized Frobenius norm of the projection of the smaller basis onto the larger basis:

$$\text{score}(r_{\text{small}}, r_{\text{large}}) = \frac{1}{r_{\text{small}}} \|U_{r_{\text{large}}}^T U_{r_{\text{small}}}\|_F^2$$

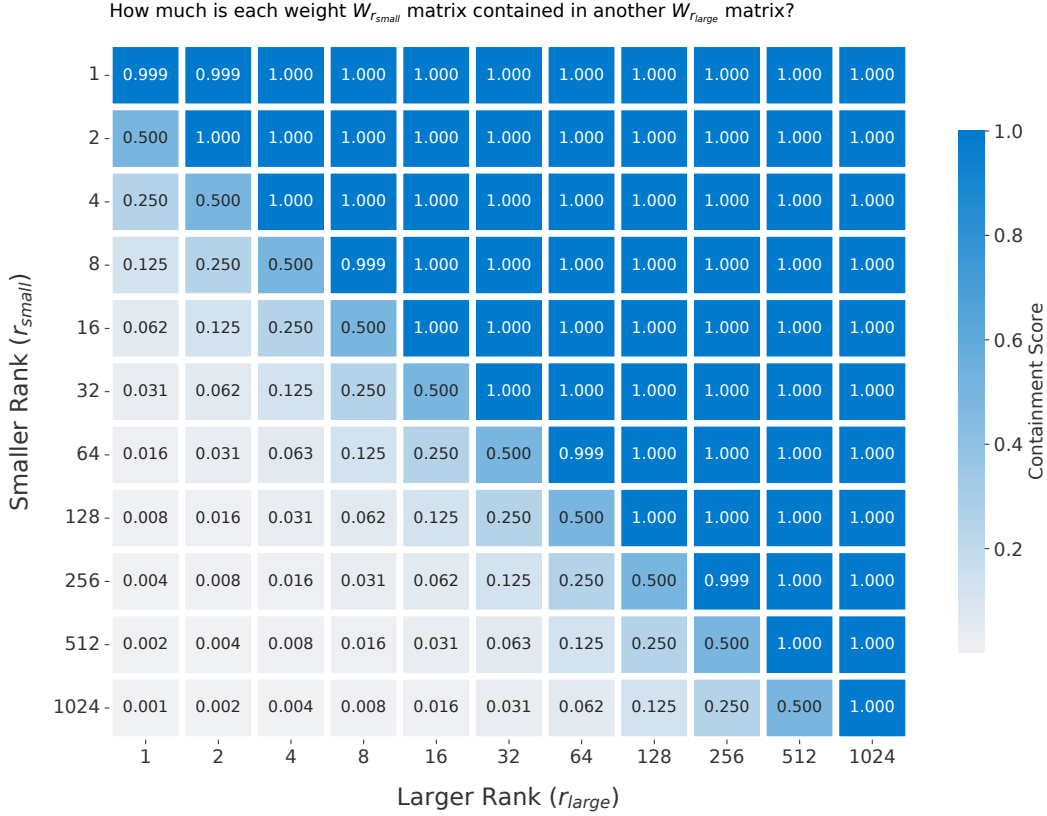
A score of 1.0 indicates that the smaller subspace is perfectly contained within the larger one.

**Results and Interpretation.** The results are visualized in the heatmap in Figure 9. The upper triangle of the matrix, where  $r_{\text{large}} \geq r_{\text{small}}$ , shows scores that are consistently 1.0 or very close to it. This empirically confirms that the vector space of a lower-rank model is indeed a nested subspace of any higher-rank model after training. The lower triangle, where  $r_{\text{large}} < r_{\text{small}}$ , shows scores significantly less than 1.0. This asymmetry is expected and acts as a sanity check, confirming that a higher-dimensional space cannot be fully contained within a lower-dimensional one.

#### A.4 CONVERGENCE ANALYSIS OF LOW-RANK VS. STANDARD FINE-TUNING

**Objective.** This experiment investigates whether a model trained with Nested Subspace layers converges to the same solution in the weight space as a model trained with standard fine-tuning. The hypothesis is that the two models will find different solutions, as the low-rank structure of NSNs acts as a form of regularization that guides the optimization process toward a different local minimum.

**Methodology.** To compare the final learned weights, a standard model was fine-tuned on the task, and a separate NSN-equipped model was trained using our proposed multi-rank objective. For the NSN model, the effective weight matrix for each layer was reconstructed at various ranks. We then computed the cosine similarity between the weight matrix of a layer from the standard fine-tuned



Note: Average score across all MLP layers. A score of 1.0 means the low-rank space is fully contained in the larger one.

Figure 9: Heatmap of subspace containment scores between weight matrices of different ranks. The score measures the extent to which the column space of a lower-rank matrix ( $r_{small}$ ) is contained within that of a higher-rank matrix ( $r_{large}$ ). A score of 1.0 (dark blue) indicates full containment. The results empirically validate the foundational nested subspace property of the trained network.

model and the corresponding reconstructed matrix from the NSN model. This comparison was performed for all MLP layers, which were grouped into early (0-10), middle (11-20), and late (21-31) stages of the network to observe depth-dependent trends.

**Results and Interpretation.** As shown in Figure 10, the weight matrices of the NSN model do not converge to the same solution as the standard fine-tuned model. The cosine similarity increases with the rank, but even at the highest rank (1024), the similarity is only around 85

This result supports the hypothesis that the nested low-rank structure imposes a regularization effect. By constraining the possible solutions to lie within pre-defined low-rank subspaces, the training process is guided to a different local minimum in the loss landscape than standard, unconstrained fine-tuning. This suggests that NSNs discover a different, yet highly effective, set of parameters for solving the task.



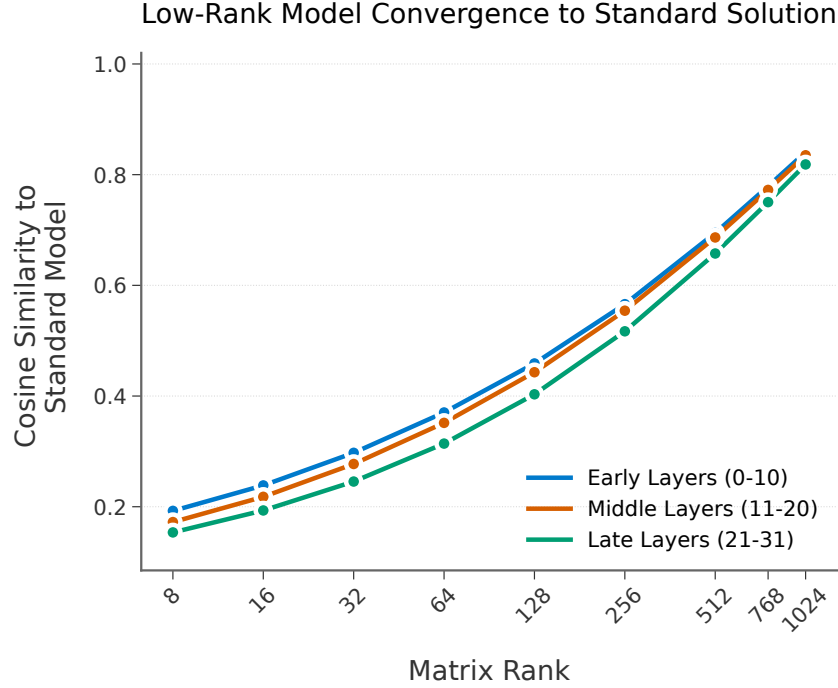


Figure 10: Cosine similarity between weight matrices from a standard fine-tuned model and a Nested Subspace Network. The similarity increases with rank but never reaches 1.0, indicating that the low-rank constraint guides the NSN to a different, yet effective, local minimum in the loss landscape.

#### A.5 COMPUTATIONAL EFFICIENCY THROUGH SURGICAL REPLACEMENT

**Objective.** This analysis demonstrates the practical computational benefits of surgically replacing standard linear layers with rank-adaptive linear layers in existing transformer architectures. The goal is to quantify the reduction in floating-point operations (FLOPs) achieved through low-rank decomposition while maintaining the nested subspace structure.

**Methodology.** We performed surgical modifications to the Gemma-2B architecture by replacing all linear layers within the MLP blocks with rank-adaptive variants. Each original weight matrix  $W$  was decomposed using Singular Value Decomposition (SVD) to initialize the factorized form  $W \approx BA$ , where  $B$  and  $A$  are lower-rank matrices. The MLP blocks, which contain three linear layers with GeLU activation functions, were systematically converted to support multiple rank configurations while preserving the original model’s functionality.

**Results and Interpretation.** The surgical replacement approach enables significant computational savings through reduced matrix operations. By decomposing the original full-rank weight matrices into their low-rank approximations, the number of parameters and corresponding FLOPs are substantially reduced. This modifica-

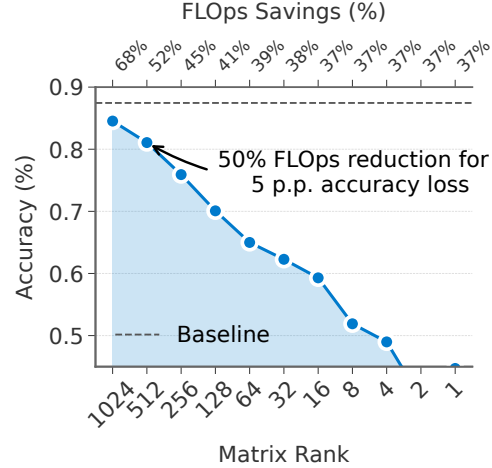


Figure 11: **Example Surgical Changes to linear layers only on Gemma-2B.** The architecture of Gemma-2B. All MLP blocks contain three linear layers with a GeLU activation function. We surgically replace all linear layers with *rank-adaptive* linear layers, initialized  $W \approx BA$  via SVD-decomposition

tion allows for dynamic rank selection during inference, providing a trade-off between computational efficiency and model capacity without requiring complete retraining of the base model.

## B ADDITIONAL THEORETICAL INSIGHTS ON NESTED SUBSPACE NETWORKS

### B.1 PROOF OF PROPOSITION

We seek to demonstrate that the performance of the network at an untrained, interpolated rank  $r_{\text{int}}$  remains close to the performance at an explicitly trained rank. This property relies on the structure induced by the training process. Let the shared, learned weight matrices be  $A \in \mathbb{R}^{R \times d_{\text{in}}}$  and  $B \in \mathbb{R}^{d_{\text{out}} \times R}$ , where  $R$  is the maximum rank. Let  $\mathbf{a}_i \in \mathbb{R}^{1 \times d_{\text{in}}}$  be the  $i$ -th row vector of  $A$  and  $\mathbf{b}_i \in \mathbb{R}^{d_{\text{out}} \times 1}$  be the  $i$ -th column vector of  $B$ .

**Lemma 1** (Adjacent Rank Perturbation). *Let  $f(\mathbf{x}; r) = (\sum_{i=1}^r \mathbf{b}_i \mathbf{a}_i) \mathbf{x}$  be the output of the linear layer for an input  $\mathbf{x}$  at rank  $r$ . The perturbation to the output when moving from rank  $r$  to  $r + 1$  is bounded by:*

$$\|f(\mathbf{x}; r + 1) - f(\mathbf{x}; r)\| \leq \|\mathbf{b}_{r+1}\| \|\mathbf{a}_{r+1}\| \|\mathbf{x}\|$$

*Proof.* The change in the weight matrix is  $W_{r+1} - W_r = \mathbf{b}_{r+1} \mathbf{a}_{r+1}$ . The change in the output is thus  $(W_{r+1} - W_r) \mathbf{x}$ . Applying the submultiplicative property of matrix and vector norms yields the result:  $\|(\mathbf{b}_{r+1} \mathbf{a}_{r+1}) \mathbf{x}\| \leq \|\mathbf{b}_{r+1} \mathbf{a}_{r+1}\| \|\mathbf{x}\| \leq \|\mathbf{b}_{r+1}\| \|\mathbf{a}_{r+1}\| \|\mathbf{x}\|$ .  $\square$

**Proposition 2** (Bound on Interpolation Error). *Let the task loss function  $\mathcal{L}(f(\mathbf{x}; r), y)$  be  $L_{\mathcal{L}}$ -Lipschitz continuous with respect to its first argument. Let  $E(r) = \mathbb{E}_{(\mathbf{x}, y)}[\mathcal{L}(f(\mathbf{x}; r), y)]$  be the expected error at rank  $r$ . For any ranks  $r_1 < r_{\text{int}} < R$ , the difference in expected error is bounded by:*

$$|E(r_{\text{int}}) - E(r_1)| \leq C \sum_{i=r_1+1}^{r_{\text{int}}} \|\mathbf{b}_i\| \|\mathbf{a}_i\|$$

where  $C = L_{\mathcal{L}} \cdot \mathbb{E}[\|\mathbf{x}\|]$  is a task-dependent constant.

*Proof.* The total change in the function output between rank  $r_1$  and  $r_{\text{int}}$  can be expressed as a telescoping sum. By the triangle inequality and Lemma 1:

$$\begin{aligned} \|f(\mathbf{x}; r_{\text{int}}) - f(\mathbf{x}; r_1)\| &= \left\| \sum_{i=r_1+1}^{r_{\text{int}}} (f(\mathbf{x}; i) - f(\mathbf{x}; i-1)) \right\| \\ &\leq \sum_{i=r_1+1}^{r_{\text{int}}} \|f(\mathbf{x}; i) - f(\mathbf{x}; i-1)\| \\ &\leq \left( \sum_{i=r_1+1}^{r_{\text{int}}} \|\mathbf{b}_i\| \|\mathbf{a}_i\| \right) \|\mathbf{x}\| \end{aligned}$$

Due to the Lipschitz continuity of the loss  $\mathcal{L}$ , the difference in expected error is bounded:

$$|E(r_{\text{int}}) - E(r_1)| \leq L_{\mathcal{L}} \cdot \mathbb{E}[\|f(\mathbf{x}; r_{\text{int}}) - f(\mathbf{x}; r_1)\|]$$

Substituting the bound on the function perturbation and defining  $C = L_{\mathcal{L}} \cdot \mathbb{E}[\|\mathbf{x}\|]$  yields the final result.  $\square$

### B.2 SVD INITIALIZATION

We propose an initialization strategy based on Singular Value Decomposition that preserves the original model parameterization at the outset of training. Formally, for a given pre-trained weight matrix  $W$ , we compute its SVD,  $W = U \Sigma V^T$ , and initialize the factor matrices  $B \in \mathbb{R}^{d_{\text{out}} \times \tilde{R}}$  and  $A \in \mathbb{R}^{\tilde{R} \times d_{\text{in}}}$  using the top  $\tilde{R}$  singular components:  $B_{\text{init}} := U_{\tilde{R}} \sqrt{\Sigma_{\tilde{R}}}$  and  $A_{\text{init}} := \sqrt{\Sigma_{\tilde{R}}} V_{\tilde{R}}^T$ , where  $U_{\tilde{R}}$  and  $V_{\tilde{R}}$  contain the first  $\tilde{R}$  columns of  $U$  and  $V$ , and  $\Sigma_{\tilde{R}}$  is the diagonal matrix of the top  $\tilde{R}$  singular values. This scheme ensures that at the maximum rank  $\tilde{R}$ , the NSN layer's effective weight matrix,  $W_{\tilde{R}} = B_{\text{init}} A_{\text{init}}$ , reconstructs the original pre-trained matrix  $W$  either exactly (if  $\tilde{R} = R$ ) or with the smallest Frobenius norm (if  $\tilde{R} < R$ ).

## C ON LLM USAGE

The authors have used large language models for three purposes:

- We have used LLMs to aid or polish our writing. This includes rephrasing text, shortening, proof-reading for ambiguities or finding mistakes or inconsistencies in notation
- We used LLMs as a supplementary source of finding related work. While we have primarily performed related work searches via google scholar, we have used the "Deep Research" functionality to find other related work that we might have missed. This has resulted in us adding response-based KD and self-distill as related work to the paper.
- We have used LLMs for research ideation early on in the paper. This included brainstorming ways how to make efficient deep neural networks, what are the properties that such neural networks should have, among others.

Otherwise, all the ideas presented in the paper are our own. We take full responsibility for any errors found in the paper.