When is Warmstarting Effective for Scaling Language Models?

Neeratyoy Mallik[1,2,*] **Maciej Janowski**[1,3,*] **Johannes Hog**[1,*]
Herilalaina Rakotoarison[4] **Josif Grabocka**[3] **Frank Hutter**[5,6,1] **Aaron Klein**[6]

[1]**University of Freiburg** [2]**Zuse School ELIZA** [3]**University of Technology Nuremberg**
[4]**University of Helsinki** [5]**Prior Labs** [6]**ELLIS Institute Tübingen**
[*]**Equal contribution.**
Correspondence: {mallik,janowski,hogj}@cs.uni-freiburg.de

Abstract

Model growth from a given checkpoint aims to accelerate training of a larger model, offering potential resource savings. Despite recent interest, warmstarting has seen limited practical adoption in large-scale training. We attribute this to two underexplored factors: (1) an overemphasis on preserving the smaller model's performance at initialization, which constrains operator design for new architectures, and (2) insufficient analysis of how growth interacts with hyperparameters and scaling behavior, compounded by inconsistent growth factors across the literature.

We show that preserving the base model's initial post-growth performance is not necessary for strong final performance, and that simple, architecture-agnostic growth strategies can outperform more complex warmstarting operators. Crucially, we empirically identify an upper bound on the growth factor g beyond which training from scratch is more efficient. We observe this across multiple ablation setups. Notably, this limit is also present, but unreported, in prior published results. Across our experiments on dense MLPs and dense language models, we find that a $2\times$ growth factor is the most reliable in yielding convergence speedups, with gains most pronounced under 20 *tokens/parameter* budgets and diminishing as budget increases. We fit scaling laws over these observations to provide guidance for practitioners deciding when and how much to grow. Together, our analysis provides practical guidelines and empirical limits for model growth.

1 Introduction

Training state-of-the-art deep learning models requires significant computational resources. As models scale from millions to billions of parameters, practitioners face an important question: can we reuse the compute invested in training smaller models when scaling up? Model growth or warmstarting (`WS`), i.e., initializing larger models from trained smaller checkpoints, offers potential efficiency gains, yet remains largely absent from production pipelines despite substantial research [Gong et al., 2019, Chen et al., 2021, Rae et al., 2021, Shen et al., 2022, Wang et al., 2023a,b, Deng et al., 2023, Samragh et al., 2024, Du et al., 2024, Yu et al., 2026].

Deep Learning (DL) success has been built on pushing the loss frontier across tasks, applications, and data modalities [Krizhevsky et al., 2012, He et al., 2016, Vaswani et al., 2017, Brown et al., 2020, Radford et al., 2021, Rombach et al., 2022, OpenAI, 2024]. As training efficacy improves, processing larger datasets drives corresponding increases in model capacity [Alabdulmohsin et al., 2023]. Current research on frontier models builds on seminal scaling law studies demonstrating the necessity of simultaneously growing both model size and data processed [Kaplan et al., 2020, Hoffmann et al., 2022]. This co-scaling question continues to evolve across different constraints, modalities, and architectures, with the traditional goal of pushing the loss-frontier now complemented by optimizing

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

the compute-optimal frontier [Anagnostidis et al., 2024]. In practice, these considerations translate to decisions about resource efficiency [Wan et al., 2023].

While WS has been periodically explored across architectures and paradigms, it has not become standard practice in training pipelines, often remaining an academic topic rather than a common choice in DL pipelines [Gong et al., 2019, Wang et al., 2023a]. We identify three barriers to practical WS adoption: 1) Implementation complexity: existing methods require architecture-specific adaptations and may not generalize to modern components [Su et al., 2021, Gu and Dao, 2023]; 2) Unclear growth limits: practitioners lack guidance on how much a given checkpoint can be grown before WS becomes counterproductive [Xie et al., 2022, Wang et al., 2023a]; 3) Hyperparameter opacity under scaling: the interaction between WS methods, hyperparameters, and model and data scaling budgets is under-explored, creating tuning overhead that undermines efficiency gains [Fetterman et al., 2023, Dey et al., 2025]. Moreover, in scaling regimes where both models and data grow to improve final convergence loss, measuring relative early-stage convergence speed may not adequately capture practical requirements [Shin et al., 2024, Unlu, 2026].

In this work, we attempt to systematically investigate when WS is truly effective, in a first-principles manner, to address these barriers. We limit the discussion scope, both for clarity and computation, for the case when a *base* model checkpoint is available to be WS-ed to a larger model scale. We assume that we are aware of the dataset the *base* was trained on, and that some of its training hyperparameters may be available, but we treat it as a *black-box* when it comes to the training compute invested in it (unlike the assumption made by Liew and Kato [2025]). We believe this better reflects a real-world scenario where only model checkpoints are publicly released. Our contributions are:

1. We generalize warmstarting growth into *shrink-zero-perturb* (SZP), a simple architecture-agnostic operation, and find that exact function-preservation is not required for effective warmstarting.
2. We empirically establish an upper bound on the growth ratio of a given checkpoint, such that, beyond this bound, training the grown model from scratch converges as fast as warmstarting it.
3. Using scaling-law fits, we show that for a given warmstarting method and base model size, the convergence advantage over training from scratch persists only within a bounded token budget, and suggest the fitted crossover as a practical guide for deciding when to warmstart.

The paper proceeds as follows: Section 2 reviews prior warmstarting methods and guidance; Section 3 introduces SZP and its design motivation; Section 4 reports hyperparameter transfer, growth limits, and scaling-law predictions; and Section 5 summarizes practical guidelines and limitations.

We release our training pipelines in the linked repository[1].

2 Related Works

Model growth as initialization is the broad class of WS techniques that derive the weights of a larger model from a smaller, typically well-trained checkpoint, before beginning training. Here, the aim is to transfer the knowledge learned by the smaller model to improve the larger model's training efficiency. Net2Net [Chen et al., 2016] and Network Morphisms [Wei et al., 2016] are among the earliest works exploring model growth for Deep Learning. At the time, the scope was limited to convolutional networks, but the core idea was carried forward to evolving architectures [Gong et al., 2019, Chen et al., 2021, Yao et al., 2024, Samragh et al., 2024]. The benefit lies in being a one-time operation before training, requiring only weights from a smaller network of the same architectural family. A central idea in designing WS operations for initialization is that of *function-preservation* (FP), wherein the grown model's output at the first forward pass matches that of the base model checkpoint used for WS. Introduced by Chen et al. [2016] for Deep Learning, FP has dominated the width-scaling WS paradigm [Wang et al., 2023b, Samragh et al., 2024, Du et al., 2024, Ma et al., 2026]. However, with new architectures, it typically requires special adaptation to remain function preserving [Chen et al., 2021, Samragh et al., 2024]. We thus question if FP at initialization is a requirement for model growth. Karp et al. [2024] already showed that loss after a short training budget correlates better than loss at initialization with growth operator rankings. We extend this to more practical training budgets.

Learned model growth is the other family of WS that does not fix the exact manner of WS operation but handles it dynamically. Wang et al. [2023a] introduces a Kronecker-factored linear operator that

[1] https://github.com/Neeratyoy/warmstarting_exps

parameterizes the linear map from the small to the larger model, learned over a few steps of training (in the style of Karp et al. [2024]). Here, the Kronecker product constrains growth ratios but works for both width and depth. Evci et al. [2022] looks at dynamic model growing during training by maximizing the gradient norms of the newly added neurons to enforce faster convergence, while the output-preservation in the *function-preservation*-style is still implied. The learned component here comes from the newly grown weights set through SVD over the gradient on the activations. This design looks to prevent dead neurons in grown models, which can lead to early rank collapse or not utilize the grown parameter capacity and move sufficiently away in the function hypothesis space from the base model. This connects to plasticity loss [Lyle et al., 2023], which correlates with dead (inactive) neurons, low effective rank representations, and growing weight magnitudes. Dohare et al. [2023] addresses this for a trained model by selectively reinitializing a fraction of low-utility neurons, scored by their activations, after each training step to keep a model in a continual learning setup. In this work, we focus entirely on the more straightforward setup of initialization-only `WS`.

Full-state `Warmstarting` looks beyond the weights. Wang et al. [2024] looks at the learning rate schedulers, Ma et al. [2026] and Yu et al. [2026] necessitate the `WS` of optimizer states too. These families aim to preserve loss (and thereby `FP`), while suitably inducing training dynamics through optimizer schedules and states, including hyperparameters. Our setup assumes that only the smaller model checkpoint is available and no other information on the optimizer or scheduler, especially their states. The framework of investigation we offer in our paper should apply to future studies extending to complete training state warmstarts, too.

`Warmstarting` **in practical settings** has been covered mostly in terms of operator selection [Karp et al., 2024], for scaling related recommendations involving scaling laws [Wang et al., 2023b, Liew and Kato, 2025], and the role of width-depth in `WS` efficacy [Du et al., 2024]. We note that most `WS` literature varies in their setups, especially with respect to growth factors g (the ratio of target to base model parameters) and the budget involved in training. Our investigation in this work focuses specifically on width-scaling strategies. Typically, joint scaling of models along both dimensions requires independent choice of width- and depth-growth operators [Du et al., 2024]. Our study here is compatible with any depth-growing and thereby joint-growing setups. We adopt a "black-box" assumption where the tokens spent on the base checkpoint are unknown, and we are concerned primarily with the question of how much we can grow this model, by training for how much budget, to ensure there is a benefit over training from scratch. In the remaining paper, we treat the terms *Model Growth* and `Warmstarting` (`WS`) synonymously.

Refer to Appendix A for additional literature on hyperparameter transfer and scaling laws.

3 Methodology

This section defines the `WS` (`Warmstarting`) setup for our empirical study (Section 4). We first introduce a compact design space for growth (Section 3.1) that abstracts common warmstarting operators. We then use this space to identify a robust, architecture-agnostic default in `SZP` (Section 3.2).

3.1 Design Space for Warmstarting

To study when `Warmstarting` is effective, we compare simple, architecture-agnostic `WS` operations against training from scratch. Function-preservation (`FP`) methods typically combine three ingredients: cloning existing weights, scaling grown weights, and adding noise, often through architecture-specific rules [Chen et al., 2016, 2021, Du et al., 2024, Samragh et al., 2024, Ma et al., 2026] (see Appendix G). We instead keep these interacting design choices independent and apply them uniformly across all components, both 1D and 2D weights.

We define our design space by extending the *shrink-and-perturb* (*SnP*) formulation from Continual Learning (CL) [Ash and Adams, 2020]. In its original form, target weights θ_{target} are derived by rescaling the base weights and adding Gaussian noise:

$$\theta^l_{\text{target}} \;=\; \lambda_{\text{shrink}} \cdot \theta^l_{\text{base}} + \mathcal{N}(0, {\sigma_{\text{perturb}}}^2) \tag{1}$$

where λ_{shrink} regulates the influence of inherited weights and σ_{perturb} controls the standard deviation of Gaussian noise, used as perturbation for all neurons.

To adapt this for model growth, where the target model is larger than the base, we augment (in blue) the formulation with a growth function, $\mathcal{G}$:

$$\theta^l_{\text{target}} \;=\; \lambda_{\text{shrink}} \cdot \underline{\mathcal{G}(\theta^l_{\text{base}}, p, q)} + \mathcal{N}(0, {\sigma_{\text{perturb}}}^2), \tag{2}$$

where $\mathcal{G} : \mathbb{R}^{m\times n} \times \mathbb{N} \times \mathbb{N} \to \mathbb{R}^{p\times q}$ maps the base weights $\theta^l_{\text{base}} \in \mathbb{R}^{m\times n}$ into the larger target shape $p \times q$ ($p \geq m, q \geq n$). Here, $\lambda_{\text{shrink}} \in \mathcal{R} > 0$ and $\sigma_{\text{perturb}} \in \mathcal{R}$, and $l \in \mathcal{N}$ denoting layer index.

Intuition and Motivation. By separating $\mathcal{G}$, λ_{shrink}, and σ_{perturb} in a `WS` operations, it gives a lightweight abstraction of common `FP`-style operations without requiring component-specific rules for exact function preservation. Formally, Equation (2) can represent more complex operators like `Net2Net` through appropriate choices of $\mathcal{G}$, λ_{shrink}, and σ_{perturb}, per architecture component [Samragh et al., 2024], though deriving such mappings remains intentionally out of scope here.

Conceptually, *Growth* determines which structure is inherited from the base model. *Perturbation* is critical for symmetry breaking, that is, it ensures that newly added neurons can learn distinct features rather than simply duplicating inherited behavior. Finally, *Shrinking* reduces the magnitude of inherited weights, dampening the dominance of base representations, and allows the new neurons to participate meaningfully in learning. Together, these axes yield simple `WS` realizations that connect cleanly to established CL concepts while remaining easy to implement. We offer theoretical intuition in Appendix C.1 and perform a mechanistic analysis of this setup in Appendix C.2.

3.2 Choosing a Default Warmstarting Method

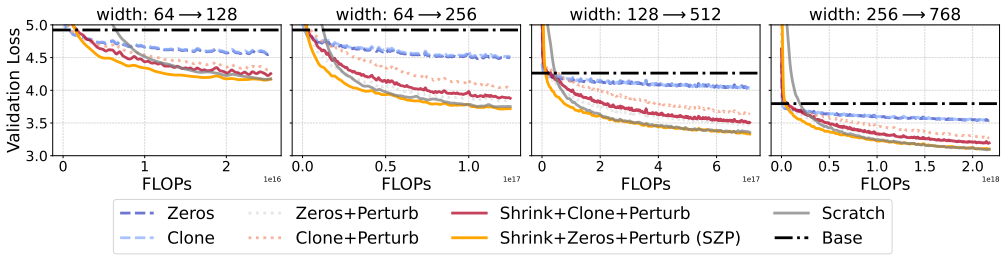


Figure 1: Comparing the simplest set of realizations of Equation (2), for different *target* scale runs, given the *base* checkpoint (*base* $\to$ *target*); `Zeros` and `Clone` represent the choices for $\mathcal{G}$, the growth function; `Zeros`+/`Clone`+`Perturb` represents the addition of noise to the grown matrix; Adding `Shrink` to the inherited base model implies a scalar multiplication of λ_{shrink} to the *base* matrix before any growth operation is applied. For either choice of $\mathcal{G}$, `Shrink`+$\mathcal{G}$+`Perturb` converges faster than $\mathcal{G}$+`Perturb`, which converges faster than $\mathcal{G}$ alone. `Scratch` runs the target model with standard initialisation (no warmstarting), while `Base` marks the final loss of the smaller base model used for `WS`. All `WS` models appear to recover and go past the `Base` loss quickly; only `SZP` keeps a convergence rate better or matched with the vanilla run from `Scratch`.

We compare representative realizations of the design space in Figure 1 to select a robust default for our scaling experiments. In Figure 1, `Zeros` and `Clone` show no significant difference in isolation, but each improves with perturbation noise (`Zeros+Perturb`, `Clone+Perturb`), and further with shrinking (`Shrink+Zeros+Perturb`, `Shrink+Clone+Perturb`) across growth ratios. We view these variants as streamlined realizations of `WS` that remain architecture-agnostic and easy to implement. A function-preservation analogue would instead set the scaling magnitude relative to grown sizes, apply a weight-type-specific cloning operation, and similarly adjust the perturbation magnitude (e.g. `Net2Net` [Chen et al., 2016], `HyperCloning` [Samragh et al., 2024]).

Notably, all `WS` variants recover the base-model loss within a few steps and converge faster than `Scratch` early in training. However, the differences between variants become clearer in sustained convergence, where `SZP` (`Shrink+Zeros+Perturb`) is the only variant consistently superior across model widths. We therefore use `SZP` as our main architecture-agnostic `WS` method.

While simplicity is subjective, `SZP` strikes a practical middle ground: it mirrors the structure of standard `FP` algorithms while loosening the strict constraint of preserving base-model outputs at initialization. We emphasize that `SZP` is not proposed as a new state-of-the-art warmstarting algorithm, but as a principled baseline we find effective for studying the scaling behavior of `WS` in Section 4.

4 Empirical Study

In this section, we use the SZP setup to study its scaling behavior, compare against popular *function-preservation* baselines such as Net2Net, observe the existence of an upper bound on the growth factor, and fit scaling laws to characterize when training from scratch outperforms Warmstarting (WS). We distill these findings into practical recommendations.

4.1 Experiment Setup

Following Qiu et al. [2025], we use the synthetic regression task as a controlled testbed for scaling and WS (refer to Appendix B for more details). This setting is computationally light to afford dense hyperparameter sweeps at both *base* and *target* widths, for all run types: Scratch, and WS runs (Net2Net and SZP). The language modeling setup, used for the broader empirical comparison, is described below. We report the corresponding compute estimates in Appendix E.

4.1.1 Hyperparameter Selection

When treating WS as an initialization, it is expected to reshape the loss landscape over hyperparameters. Prior comparisons of WS operations have under-explored this. This matters particularly when WS to much larger models, where re-tuning costs can be prohibitive. Here, we use the synthetic MLP regression task to compare tuning strategies across WS operations.

Training Hyperparameters. The tuned hyperparameters are detailed in Table 2 and summarized in Figure 2, where violins show the final validation loss distribution from a dense grid search at each target width for Scratch, Net2Net, and SZP. For transfer, we take the best configuration at the base width of 48 and transfer it as (1) unchanged (*static transfer*), or (2) with width-aware μP scaling rules [Yang et al., 2021]. We refer readers to Appendix B.4 for background and details on μP.

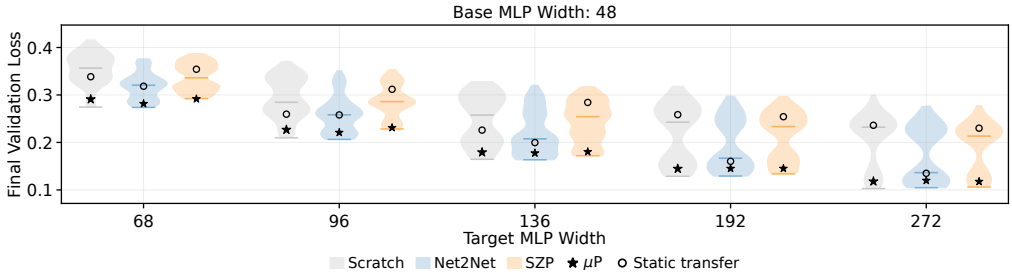


Figure 2: Final validation loss under MLP-width scaling from a fixed base width of 48. For each *target* width, each triplet of violins shows Scratch, Net2Net, and SZP from left to right. Violins summarize the final-loss distribution over a dense target-scale hyperparameter grid (Appendix, Table 2); horizontal ticks denote the median and minimum losses within each grid. Open circles mark **static transfer**: the best base-width hyperparameters unchanged at the target width. Stars mark **width-aware transfer**: the base-width optimum follows μP scaling rules (Appendix B.5).

Figure 2 shows that dense tuning at the target width usually yields the *best* configuration when the budget allows. The effectiveness of transfer, however, depends on the initialization method. Static transfer worsens with growth factor for both Scratch and SZP, while Net2Net's *function-preservation* aspect makes it more robust. Crucially, μP transfer is consistently near-optimal relative to the dense grid and outperforms static transfer across all run types. This motivates us to use μP transfer for all subsequent compute-heavy experiments, especially the language model runs reported in later sections. Notably, the best attainable losses across dense grid search over Scratch, Net2Net, and SZP converge as the growth factor increases. This hints at the practical limits of WS that we examine more closely in Section 4.2.

SZP **Design Choices.** In *function-preservation*-based methods, the perturbation noise and scaling applied to base or new neurons depend on the architectural component. We instead apply Equation (2) uniformly across all weights and thus fix the scaling variable and the perturbation noise form. For perturbation noise, we use the scale-dependent $\sigma_{\text{perturb}} = 1/\sqrt{\text{width}}$ applied under both Static

transfer and μP. This choice is common in Deep Learning and has also been used in FP-based setups [Ma et al., 2026]. We provide an ablation on the perturbation noise in Appendix C.3, where we observe that the noise scale affects the convergence rate more than the final generalization loss.

Just as perturbation is essential for new neurons to break symmetry inherited from the base model and use the full target capacity [Yu et al., 2026], shrinking is essential for reducing weight norms, increasing effective ranks, and maintaining plasticity. We follow Ash and Adams [2020] and fix $\lambda_{\text{shrink}} = 0.4$ in our experiments to isolate the effect of other variables. Our ablation in Appendix C.4 confirms that $\lambda_{\text{shrink}} \in \{0.2,\ 0.4\}$ offers improved performance over training from scratch for reasonable growth factors. We additionally provide a simple theoretical sketch in Appendix C.1 for why zero-padding, perturbation, and shrinkage play complementary roles.

4.1.2 Language Model Setup

All language-model experiments train decoder-only GPT-2 style models [Radford et al., 2019] using the open-source LitGPT implementation [LitGPT, 2023] on the SlimPajama corpus [Soboleva et al., 2023] (Figure 1 uses sampled version[2]). We perform only width-scaling here, consistent with our scope of studying FP-based WS. All runs used AdamW and a warmup-stable-decay learning rate scheduler with warmup and cooldown lengths set to fixed fractions of the total training budget. Each run's training budget is set to $10,\ 20,$ or 30 *tokens/parameter*. We train models ranging from 14M to 1.2B parameters. Except for dense grid searches at multiple base scales, all `Scratch` and WS runs at scales larger than the base use μP transfer for learning rates, and we scale the base-scale optimal batch size. Additional training and hyperparameter details are provided in Appendix D. We additionally present ablations of SZP performance on the LLaMa architecture in Appendix F.1.

4.2 Practical `Warmstarting`

In this section, we focus on language model experiments and address our primary question of how and where to find gains when `Warmstarting`.

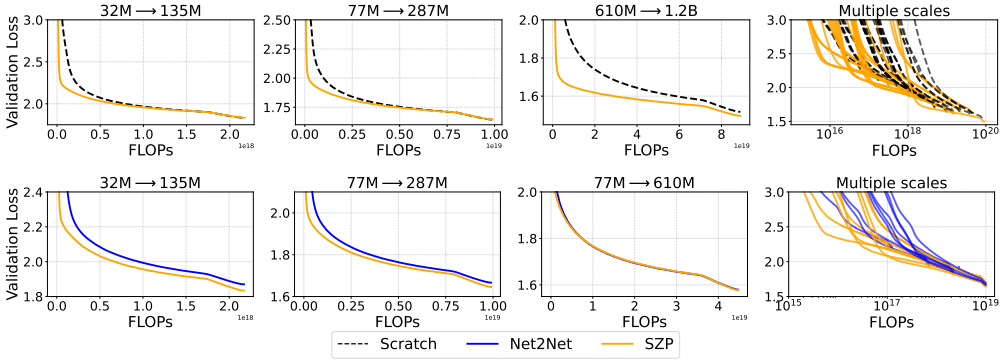


Figure 3: Warmstarting from a *base→target* model size, trained for 20 *tokens/parameter*. *Top row*: comparing `Scratch` to SZP up to 1.2B parameters, showing that SZP converges faster and achieves better final loss, with gains varying across growth factors g (rightmost). *Bottom row*: comparing `Net2Net` to SZP (both WS methods explored here) for up to 610M parameters, showing that SZP consistently outperforms the FP-based baseline.

Empirical Performance of SZP. We report performance scaling for SZP up to 1.2B parameters in Figure 3 (top), including a single-checkpoint downstream sanity check in Appendix F.2. Both rows show that SZP converges faster than `Scratch` and `Net2Net` (for `HyperCloning` results see Appendix G.5)). Most notably, the relative performance gains of the WS operations (`Net2Net` and SZP) diminish as the growth factor g grows. Figure 3 (rightmost, both rows) shows all the learning curves trained for $10,\ 20,$ and 30 *tokens/parameter*s, for different base-target pairings, with identical learning rate schedules, confirming that faster convergence holds across scales for both comparisons. The difference in final loss is more subtle and is discussed more in later sections, where we fit scaling

[2] `https://huggingface.co/datasets/DKYoon/SlimPajama-6B`

laws and systematically characterize the regimes in which WS outperforms training from Scratch, and SZP outperforms Net2Net.

The variance in performance difference across growth factors is a central observation of this work. The base model used to derive grown weights is typically well-trained, with its learned function occupying a subspace of the weight space, and exploiting this subspace is where WS's efficiency gains originate (signal preservation). For *function-preservation* methods, which typically rely on *clone* operations, this subspace can be hard to escape due to inherent symmetries in the grown weights [Yu et al., 2026]. Figure 8 (right) (see, Appendix C.2) reflects this in the lower effective rank of Net2Net compared to SZP. Decomposing weight movement into radial (along the initialization) and angular (orthogonal) components, we find that SZP's attention layers show both rotation and scaling of the weights at initialization, while Net2Net's radial component remains much more stable and weight adjustment is dominated by rotations than scaling the initialization vector. The practical impact of this may show up in too well-trained base models saturating WS performance [Liew and Kato, 2025] or growing a base checkpoint too large such that gains diminish. In the next section, we observe and discuss this in more detail.

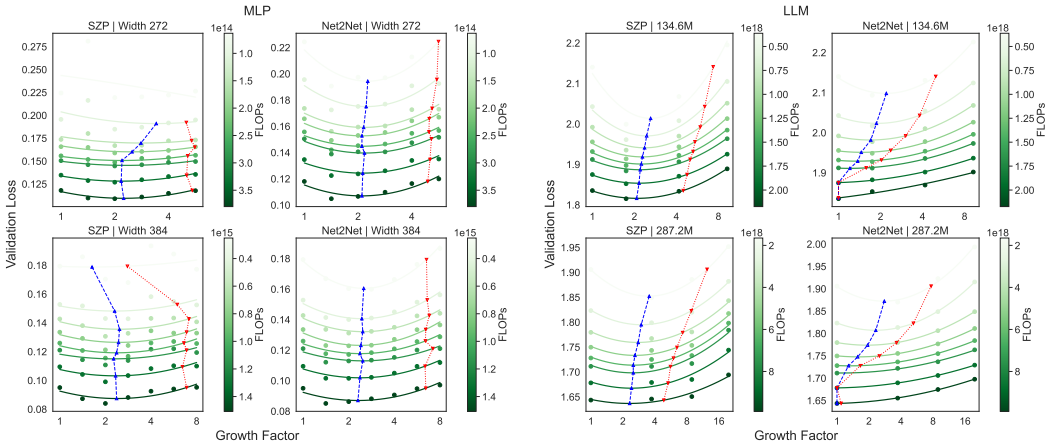


Figure 4: IsoFLOPs on Language Models and MLPs for 20 *tokens/parameter*. Blue lines (the line connecting the trough of the IsoFLOPs) mark the optimal growth factor, g_{opt}, at each IsoFLOP line, while red lines (the vertical line right of the g_{opt} line) mark the g_{upper}, the upper bound on the growth factor at which WS matches the loss of training from Scratch ($g = 1$). These lines are obtained from a quadratic fit to the validation loss rather than directly from the loss values. Additional results for more target sizes and budgets can be found in Appendix H.1.

The Bounded Effective Growth Factor. For practitioners with a fixed base checkpoint seeking to scale up (as scoped in Section 1), a key question is how much the model can grow while preserving WS's efficiency gains over Scratch. Prior work [Du et al., 2024] shows that WS performance depends on the growth factor g, defined as the ratio of target to base model parameters, and characterizes this via the optimal growth factor. This optimal factor, which we term g_{opt}, is estimated by fitting quadratic models to IsoFLOP losses and minimizing them. g_{opt} is meant to be an *optimal* growth factor such that it maximizes the relative gains of WS over training from Scratch. We argue this optimization-based characterization is hard to support from sparse evaluations, since reliably identifying g_{opt} requires substantially denser measurements, especially for $g \in (1, g_{\text{opt}})$. Instead, we propose reporting g_{upper}, which we observe as the largest growth factor under which WS still improves over Scratch's final validation loss. This can be extracted by drawing a horizontal line from $g = 1$ for each IsoFLOP line, cutting the parabola at a $g > 1$. We posit that g_{upper} offers a more flexible and practical recommendation for WS in practice, and assess relative gains of different WS operations.

While such a bound is implicit in prior results Du et al. [2024], it has not been reported as a quantity of interest. In Figure 4, we reproduce these results for SZP and Net2Net in the LLM setting (MLP included as verification), modeling IsoFLOP lines with quadratic fits across growth factors ($g = 1$ denotes Scratch). Both g_{opt} (blue) and g_{upper} (red) are read off the fit. Bootstrap uncertainty estimates for both quantities are provided in Appendix H.1. g_{upper} is observed across all settings. The exception is Net2Net in the LLM setting, where $g_{\text{upper}} = 1$ (the Scratch run) at the final IsoFLOPs eventually

beats `Net2Net` by the end of the scheduled budget (when including the learning rate decay). This is consistent with Du et al. [2024][3] and highlights the shortcomings of function-preservation methods on more complex architectures (refer to Appendix G). For complete IsoFLOPs results and an additional analysis accounting for base-model training compute, refer to Appendix H.1.

One interpretation is that `Warmstarting` places the target model in a region of the loss landscape shaped by the base model's learned representations. For small g, this initialization provides a favorable basin for efficient convergence. As g grows, the base model occupies a diminishing fraction of the target architecture, and the initialization increasingly resembles a random one. Beyond g_{upper}, the constraints imposed by the inherited structure outweigh its benefits, anchoring the warmstarted model in a suboptimal region. Consistent with this, our mechanistic analysis (Figure 13) shows that warmstarted models at higher g increasingly resemble `Scratch` across interpretability metrics, suggesting gradual dilution of the base model's structure. The stability of g_{upper} across model scales further supports this. The rate at which warmstarted representations converge to `Scratch`-like behavior depends primarily on the architectural ratio g, and not on absolute model size.

In summary, we observe a bound on the effective growth factor g_{upper}, consistent with Du et al. [2024], and argue that identifying and reporting it is crucial for practical adoption of `WS`. When budget does not permit investigating g_{upper}, we recommend $g = 2$ as a conservative default. Much of the `WS` literature restricts to $2\times$ growth [Samragh et al., 2024, Ma et al., 2026], leaving g_{upper} undiscussed. For fair comparison of `WS` operations, g_{upper} should be discussed and, where possible, investigated.

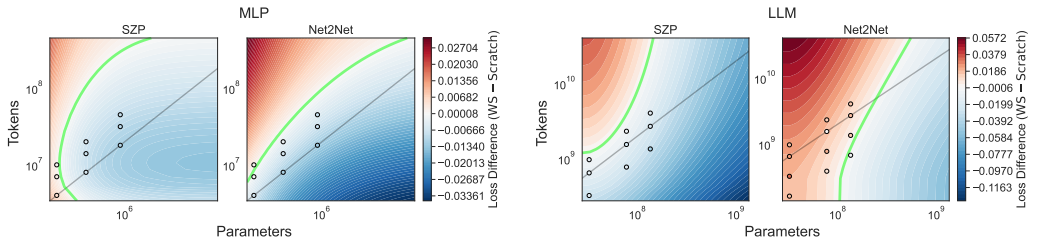


Figure 5: Isolines for *loss difference* showing where `Warmstarting` achieves lower loss than training from `Scratch` for growth factor $g \approx 2$ (blue regions indicate `Warmstarting` is better; red indicates `Scratch` is better; the green line marks the boundary where both perform equally). Given a fixed model size, the more the *tokens/parameter*, the more preferred is training from `Scratch` over `Warmstarting`. The region where `SZP` is effective is larger than that of Net2Net, enabling a more complete comparison between growth methods.

Effective Warmstarting Across Scale. Scaling a language model involves growing both the model and the data budget. As discussed, diminishing gains are expected from `WS`, so studying its effectiveness at a single token budget and sparse model sizes provides only a limited, possibly misleading, signal. While Liew and Kato [2025] identified that overtraining base models diminishes `WS` gains, we highlight the effect of g_{upper} on parameter scaling. Since we assume the base model's training budget is unknown, we focus on the practical requirement of specifying the target budget.

We subset our collected data for $g \approx 2$ (Figure 4) and follow Approach 3 from Hoffmann et al. [2022] to fit the power-law parametric form predicting loss as a function of parameters N and tokens D,

$$L(N, D) = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}},$$

where E, A, α, B, β are fit by optimizing the Huber loss ($\delta = 10^{-3}$) on $\log L$ with L-BFGS-B. We fit scaling laws for `Scratch`, `SZP`, and `Net2Net` in both LLM and MLP settings. Appendix H.2 reports the parameters and R^2 values, all above 0.99, confirming fit quality. In Figure 5, we visualize the *difference* in modeled loss between `WS` methods and `Scratch` fits as *Isolines* (contour lines of equal loss difference), marking the zero-difference contour in green as the boundary for our analysis. First, the existence of this boundary across MLPs and LLMs indicates regimes where training from `Scratch` can be preferred over `WS`. Second, the boundary's position relative to the Chinchilla-optimal

[3] Du et al. [2024] reports that width-only scaling with `FP` underperforms, while their IsoFLOP analysis on growth factors for width-only scaling also exhibits a g_{upper} bound.

tokens/parameter front (solid black line) suggests that training for fewer tokens/parameter is more likely to retain WS gains. Importantly, this bound provides practical guidance, subject to the quality of the fits.

These scaling-law fits and *Iso-Loss Difference* contours provide a wider picture that corroborates our initial hypothesis on the necessity of *function-preservation* for WS. In the MLP setup, Net2Net provides a clear front of WS gains with stronger final performance than Scratch. This is an expected result, as *function-preservation* was designed for MLPs. However, SZP provides a larger range of WS efficacy, with the loss-difference front pushed further from the Chinchilla-optimal front. As a contrasting case, for LLMs we observe that Net2Net shows only a small region of predicted gains over Scratch, consistent with the trends in Figure 4. For LLMs, SZP offers a reasonable front, indicating scaling trends for when WS becomes worse than Scratch training.

In summary, shorter training budgets (≤ 20 *tokens/parameter*) are more likely to yield WS gains over Scratch. For fair comparison of WS operations, the scaling pattern must be assessed and the front beyond which WS loses its advantage characterized, yielding practical guidance.

5 Conclusion

We have characterized the upper bound on the growth factor within which Warmstarting (WS) remains computationally advantageous over training from Scratch. Beyond g_{upper}, efficiency gains diminish significantly. Our analysis reveals that this limit is fundamentally tied to the training budget. We distill our contributions into a set of principled guidelines for the community:

- **Model Scaling:** If only one model run is feasible, $g = 2$ is the most reliable scaling for WS.
- **Efficiency Window:** WS should be prioritized over Scratch when training at or below compute-optimal scaling (20 *tokens/parameter*).
- **LR Transfer:** μP serves as a principled strategy for hyperparameter transfer in WS, particularly when tuning budgets for larger scales are constrained.
- **Standardized Reporting:** We propose future WS literature explicitly report the empirical g_{upper} wherever possible, and that comparisons account for scaling patterns of diminishing WS gains.

We show that SZP (*Shrink-Zero-Perturb*), which augments zero-padding with a *shrink-and-perturb* strategy, is architecture-agnostic and more broadly applicable than function-preserving methods. SZP is easier to interpret, control, and study across architectures and setups, and our work here offers future WS literature both a framework to assess limits and make fairer comparisons of WS operations.

Limitations. Our empirical study is constrained by the computational costs of language model training and the need for controlled comparisons. To reflect realistic scenarios where only model checkpoints are available, we treat the base model as a black box regarding its training history: SZP needs only the weight checkpoint, not optimizer states or schedule information [Shen et al., 2022]. While this distinguishes our work from studies that assume full access to base training compute [Liew and Kato, 2025], it limits our ability to analyze the joint compute-optimal frontier across base size, target size, and tokens invested in both training phases. However, our findings at sub-1B scales align with trends reported at 1B+ scales in prior work, suggesting transferability to larger regimes. Additionally, exhaustive hyperparameter sweeps across all growth operators remain computationally prohibitive at the language model scale. Our comparison focuses on SZP vs. Net2Net to establish a representative baseline, with the understanding that SZP can serve as a modular alternative for width-scaling methods extending Net2Net. Especially, combining with depth-scaling [Du et al., 2024] for jointly growing both depth and width.

Future Directions. Several promising avenues emerge from this work. First, benchmarking SZP across diverse architectures and data modalities would validate its generality beyond language models. Second, understanding the individual contributions of shrinking and perturbation within *shrink-and-perturb* could reveal whether these components can be predictively scaled using fixed rules or scaling law fits, and extending to WS of optimizers too. Third, given the existence of upper bounds on growth factors, exploring progressive growing and optimal growth schedules becomes a natural next step. Fourth, extending the scaling law to jointly model loss as a function of parameters, tokens, and growth factor would remove the need to fix the growth factor across experiments, enabling predictions for

arbitrary growth factors from a single unified fit. More broadly, as specialized model checkpoints become increasingly available, strengthening the practical impact of warmstarting methods will be critical for improving resource efficiency across the field.

Acknowledgements

We thank Tarek Abou Chakra and Samir Garibov for their code support in the initial days of the project. We thank André Biedenkapp for feedback on our paper.

FH acknowledges the financial support of the Hector Foundation. **NM**, **JH**, **HR**, **FH** acknowledge funding by the state of Baden-Württemberg through bwHPC, the German Research Foundation (DFG) through grant numbers INST 39/963-1 FUGG and 417962828, and the European Union (via ERC Consolidator Grant Deep Learning 2.0, grant no. 101045765). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. This research was also funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 539134284, through EFRE (FEIH_2698644) and the state of Baden-Württemberg. **NM** is supported by the Konrad Zuse School of Excellence in Learning and Intelligent Systems (ELIZA) through the DAAD programme Konrad Zuse Schools of Excellence in Artificial Intelligence, sponsored by the Federal Ministry of Education and Research. **JH** and **FH** acknowledge funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under SFB 1597 (SmallData), grant number 499552394. **MJ** and **JG** acknowledge funding by The Carl Zeiss Foundation through the research network "Responsive and Scalable Learning for Robots Assisting Humans" (ReScaLe) of the University of Freiburg. **AK** acknowledges support from EC under the grant No. 101195233 (OpenEuroLLM).



References

I. Alabdulmohsin, X. Zhai, A. Kolesnikov, and L. Beyer. Getting Vit in Shape: Scaling laws for compute-optimal model design. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Proceedings of the 36th International Conference on Advances in Neural Information Processing Systems (NeurIPS'23)*, 2023.

S. Anagnostidis, G. Bachmann, I. Schlag, and T. Hofmann. Navigating scaling laws: Compute optimality in adaptive model training. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning (ICML'24)*, volume 251 of *Proceedings of Machine Learning Research*. PMLR, 2024.

J. Ash and R. P. Adams. On warm-starting neural network training. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H. Lin, editors, *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*, 2020.

S. Bergsma, B. C. Zhang, N. Dey, S. Muhammad, G. Gosal, and J. Hestness. Scaling with collapse: Efficient and predictable training of llm families. *arXiv preprint arXiv:2509.25087*, 2025.

Y. Bisk, R. Zellers, R. Le Bras, J. Gao, and Y. Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

J. Bjorck, A. Benhaim, V. Chaudhary, F. Wei, and X. Song. Scaling optimal lr across token horizons. 2025.

C. Blake, C. Eichenberg, J. Dean, L. Balles, L. Y. Prince, B. Deiseroth, A. F. Cruz-Salinas, C. Luschi, S. Weinbach, and D. Orr. u-μP: The unit-scaled maximal update parametrization. In *The Thirteenth International Conference on Learning Representations (ICLR'25)*. ICLR, 2025.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H. Lin, editors, *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*, pages 1877–1901, 2020.

E. Caballero, K. Gupta, I. Rish, and D. Krueger. Broken neural scaling laws. In *The Eleventh International Conference on Learning Representations (ICLR'23)*. ICLR, 2023.

C. Chen, Y. Yin, L. Shang, X. Jiang, Y. Qin, F. Wang, Z. Wang, X. Chen, Z. Liu, and Q. Liu. bert2bert: Towards reusable pretrained language models. In S. Muresan, P. Nakov, and A. Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2134–2148. Association for Computational Linguistics, 2021.

T. Chen, I. Goodfellow, and J. Shlens. Net2net: Accelerating learning via knowledge transfer. In *The Fourth International Conference on Learning Representations (ICLR'16)*. ICLR, 2016.

L. Clare. Does parameter norm growth cause networks to lose plasticity? Clare Lyle's Blog, June 2025. URL `https://clarelyle.com/posts/2025-06-30-plasticity-norm.html`.

P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

X. Deng, H. Shi, R. Huang, C. Li, H. Xu, J. Han, J. Kwok, S. Zhao, W. Zhang, and X. Liang. Growclip: Data-aware automatic model growing for large-scale contrastive language-image pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 22178–22189, 2023.

N. Dey, S. Bergsma, and J. Hestness. Sparse maximal update parameterization: A holistic approach to sparse training dynamics. *arXiv preprint arXiv:2405.15743*, 2024.

N. Dey, B. C. Zhang, L. Noci, M. Li, B. Bordelon, S. Bergsma, C. Pehlevan, B. Hanin, and J. Hestness. Don't be lazy: CompleteP enables compute-efficient deep transformers. *arXiv preprint arXiv:2505.01618*, 2025.

S. Dohare, J. F. Hernandez-Garcia, P. Rahman, A. R. Mahmood, and R. S. Sutton. Maintaining plasticity in deep continual learning. *arXiv preprint arXiv:2306.13812*, 2023.

W. Du, T. Luo, Z. Qiu, Z. Huang, Y. Shen, R. Cheng, Y. Guo, and J. Fu. Stacking your transformers: A closer look at model growth for efficient llm pre-training. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Proceedings of the 37th International Conference on Advances in Neural Information Processing Systems (NeurIPS'24)*, 2024.

M. Elsayed, Q. Lan, C. Lyle, and A. R. Mahmood. Weight clipping for deep continual and reinforcement learning. In *Proceedings of the Reinforcement Learning Conference (RLC)*, pages 2198–2217, 2024.

Essential AI et al. Practical efficiency of muon for pretraining. *arXiv preprint arXiv:2505.02222*, 2025.

U. Evci, B. van Merrienboer, T. Unterthiner, M. Vladymyrov, and F. Pedregosa. Gradmax: Growing neural networks using gradient information. In *The Tenth International Conference on Learning Representations (ICLR'22)*. ICLR, 2022.

K. E. Everett, L. Xiao, M. Wortsman, A. A. Alemi, R. Novak, P. J. Liu, I. Gur, J. Sohl-Dickstein, L. P. Kaelbling, J. Lee, and J. Pennington. Scaling exponents across parameterizations and optimizers. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning (ICML'24)*, volume 251 of *Proceedings of Machine Learning Research*. PMLR, 2024.

A. J. Fetterman, E. Kitanidis, J. Albrecht, Z. Polizzi, B. Fogelman, M. Knutins, B. Wróblewski, J. B. Simon, and K. Qiu. Tune as you scale: Hyperparameter optimization for compute efficient training. *arXiv preprint arXiv:2306.08055*, 2023.

O. Filatov, J. Wang, J. Ebert, and S. Kesselheim. Optimal scaling needs optimal norm. *arXiv preprint arXiv:2510.03871*, 2025.

L. Gong, D. He, Z. Li, T. Qin, L. Wang, and T. Liu. Efficient training of BERT by progressively stacking. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*, volume 97, pages 2337–2346. Proceedings of Machine Learning Research, 2019.

A. Gu and T. Dao. Mamba: Linear time sequence modeling with selective state spaces. *arXiv:2312.00752 [cs.LG]*, 2023.

N. Habib, C. Fourrier, H. Kydlíček, T. Wolf, and L. Tunstall. Lighteval: A lightweight framework for llm evaluation, 2023. URL `https://github.com/huggingface/lighteval`.

A. Hägele, E. Bakouch, A. Kosson, L. Ben allal, L. Von Werra, and M. Jaggi. Scaling laws and compute-optimal training beyond fixed training durations. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Proceedings of the 37th International Conference on Advances in Neural Information Processing Systems (NeurIPS'24)*, 2024.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'16)*, pages 770–778, 2016.

D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre. Training compute-optimal large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Proceedings of the 35th International Conference on Advances in Neural Information Processing Systems (NeurIPS'22)*, 2022.

G. Hooker. Generalized functional ANOVA diagnostics for high-dimensional functions of dependent variables. *Journal of Computational and Graphical Statistics*, 16:709–732, 2007.

Hugging Face. Open llm leaderboard v1. `https://huggingface.co/docs/leaderboards/open_llm_leaderboard/archive`, 2024a. Archived Open LLM Leaderboard documentation.

Hugging Face. LightEval open llm leaderboard tasks. `https://github.com/huggingface/lighteval/blob/main/examples/tasks/open_llm_leaderboard_tasks.txt`, 2024b. Task configuration file specifying leaderboard tasks and few-shot settings.

A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'18)*, volume 31, 2018.

J. Kaplan, S. McCandlish, T. Henighan, T. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv:2001.08361 [cs.LG]*, 2020.

S. Karp, N. Saunshi, S. Miryoosefi, S. J. Reddi, and S. Kumar. Landscape-aware growing: The power of a little lag. *arXiv preprint arXiv:2406.02469*, 2024.

A. Kosson, B. Messmer, and M. Jaggi. Analyzing & reducing the need for learning rate warmup in gpt training. *Advances in Neural Information Processing Systems*, 37:2914–2942, 2024.

A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NeurIPS'12)*, pages 1097–1105, 2012.

S. P. Liew and T. Kato. From acceleration to saturation: Scaling behavior of bootstrapped language model pretraining. *arXiv preprint arXiv:2510.06548*, 2025.

LitGPT. Lightning AI. `https://github.com/Lightning-AI/litgpt`, 2023.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *The Seventh International Conference on Learning Representations (ICLR'19)*. ICLR, 2019.

C. Lyle, Z. Zheng, E. Nikishin, B. A. Pires, R. Pascanu, and W. Dabney. Understanding plasticity in neural networks. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*, volume 202 of *Proceedings of Machine Learning Research*, pages 23190–23211. PMLR, 2023.

Y. Ma, N. Chen, M. Díaz, S. Hayou, D. Kunisky, and S. Villar. μpscaling small models: Principled warm starts and hyperparameter transfer. *arXiv preprint arXiv:2602.10545*, 2026.

T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, 2018.

OpenAI. Gpt-4 technical report, 2024. URL `https://arxiv.org/abs/2303.08774`.

T. Pethick, W. Xie, K. Antonakopoulos, Z. Zhu, A. Silveti-Falls, and V. Cevher. Training deep learning models with norm-constrained lmos. In A. Singh, M. Fazel, D. Hsu, S. Lacoste-Julien, F. Berkenkamp, T. Maharaj, K. Wagstaff, and J. Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning (ICML'25)*, volume 267 of *Proceedings of Machine Learning Research*. PMLR, 2025.

R. Porian, M. Wortsman, J. Jitsev, L. Schmidt, and Y. Carmon. Resolving discrepancies in compute-optimal scaling of language models. In *2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ICML 2024)*, 2024.

Y. Qin, Y. Lin, J. Yi, J. Zhang, X. Han, Z. Zhang, Y. Su, Z. Liu, P. Li, M. Sun, et al. Knowledge inheritance for pre-trained language models. *arXiv preprint arXiv:2105.13880*, 2021.

S. Qiu, L. Xiao, A. G. Wilson, J. Pennington, and A. Agarwala. Scaling collapse reveals universal dynamics in compute-optimally trained neural networks. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 50697–50720. PMLR, 2025.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, and J. Clark. Learning transferable visual models from natural language supervision. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 2021.

J. W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'22)*, pages 10684–10695, 2022.

K. Sakaguchi, R. Le Bras, C. Bhagavatula, and Y. Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

M. Samragh, I. Mirzadeh, K. A. Vahid, F. Faghri, M. Cho, M. Nabi, D. Naik, and M. Farajtabar. Scaling Smart: Accelerating large language model pre-training with small model initialization. *arXiv:2409.12903 [cs.CL]*, 2024.

S. Shen, P. Walsh, K. Keutzer, J. Dodge, M. Peters, and I. Beltagy. Staged training for transformer language models. In *International Conference on Machine Learning*, pages 19893–19908. PMLR, 2022.

B. Shin, J. Oh, H. Cho, and C. Yun. Dash: Warm-starting neural network training without loss of plasticity under stationarity. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ ICML 2024)*, volume 251 of *Proceedings of Machine Learning Research*. PMLR, 2024.

D. Soboleva, F. Al-Khateeb, R. Myers, J. R. Steeves, J. Hestness, and N. Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama, June 2023. URL `https://huggingface.co/datasets/cerebras/SlimPajama-627B`.

B. Sorscher, R. Geirhos, S. Shekhar, S. Ganguli, and A. S. Morcos. Beyond neural scaling laws: beating power law scaling via data pruning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Proceedings of the 35th International Conference on Advances in Neural Information Processing Systems (NeurIPS'22)*, 2022.

J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv:2104.09864 [cs.CL]*, 2021.

B. Thérien, C. Joseph, B. Knyazev, E. Oyallon, I. Rish, and E. Belilovsky. μlo: Compute-efficient meta-generalization of learned optimizers. *arXiv preprint arXiv:2406.00153*, 2024.

H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

E. Unlu. Preservation is not enough for width growth: Regime-sensitive selection of dense lm warm starts. *arXiv preprint arXiv:2604.04281*, 2026.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'17)*. Curran Associates, Inc., 2017.

N. Vyas, A. Atanasov, B. Bordelon, D. Morwani, S. Sainathan, and C. Pehlevan. Feature-learning networks are consistent across widths at realistic scales. 36:1036–1060, 2023.

Z. Wan, X. Wang, C. Liu, S. Alam, Y. Zheng, et al. Efficient large language models: A survey. *arXiv preprint arXiv:2312.03863*, 1, 2023.

P. Wang, R. Panda, L. T. Hennigen, P. Greengard, L. Karlinsky, R. Feris, D. D. Cox, Z. Wang, and Y. Kim. Learning to grow pretrained models for efficient transformer training. In *The Eleventh International Conference on Learning Representations (ICLR'23)*. ICLR, 2023a.

P. Wang, R. Panda, and Z. Wang. Data efficient neural scaling law via model reusing. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*, volume 202 of *Proceedings of Machine Learning Research*, pages 36193–36204. PMLR, 2023b.

Y. Wang, J. Su, H. Lu, C. Xie, T. Liu, J. Yuan, H. Lin, R. Sun, and H. Yang. Lemon: Lossless model expansion. In *The Twelfth International Conference on Learning Representations (ICLR'24)*. ICLR, 2024.

T. Wei, C. Wang, Y. Rui, and C. W. Chen. Network morphism. In M. Balcan and K. Weinberger, editors, *Proceedings of the 33rd International Conference on Machine Learning (ICML'17)*, volume 48, pages 564–572. Proceedings of Machine Learning Research, 2016.

J. Xie, X. Su, S. You, Z. Ma, F. Wang, and C. Qian. Scalenet: Searching for the model to scale. In S. Avidan, G. Brostow, G. Farinella, and T. Hassner, editors, *17th European Conference on Computer Vision (ECCV'22)*, pages 104–120, 2022.

G. Yang and E. J Hu. Tensor programs IV: Feature learning in infinite-width neural networks. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*, volume 139 of *Proceedings of Machine Learning Research*. PMLR, 2021.

G. Yang, E. Hu, I. Babuschkin, S. Sidor, X. Liu, D. Farhi, N. Ryder, J. Pachocki, W. Chen, and J. Gao. Tuning large neural networks via zero-shot hyperparameter transfer. In M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin, editors, *Proceedings of the 34th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*, 2021.

Y. Yao, Z. Zhang, J. Li, and Y. Wang. Masked structural growth for 2x faster language model pre-training. In *The Twelfth International Conference on Learning Representations (ICLR'24)*. ICLR, 2024.

Q. Yu, X. Ma, Z. Zhuo, M. Wang, D. Liu, S. Zhan, Y. Ma, L. Xiang, X. Bin, and D. He. Sparkling: Balancing signal preservation and symmetry breaking for width-progressive learning. *arXiv preprint arXiv:2602.02472*, 2026.

R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, 2019.

A Supporting Related Work

Scaled parameterizations refer to a set of rules that determine how certain parameters can be scaled with respect to one or more scaling dimensions [Everett et al., 2024]. This family of parameterizations describes scaling factors for the weights, the learning rate, and the standard deviation of the initialization. Each of these scaling variables can be defined separately for the input, hidden, and output layers (see Tables 1, 2, 3[4] in Yang and Hu [2021]). The maximal update parameterization or μP is one such parameterization that showed its scaling rules to allow for stable training of larger networks without requiring re-tuning of the variables (such as learning rate) being scaled [Yang et al., 2021], i.e., feature learning does occur at infinite-width scaling [Yang and Hu, 2021], extending the Neural Tangent Kernel [Jacot et al., 2018] theory demonstrating lazy-training (fixed features) when the network is scaled to infinite-width. Follow-up works exploited the symmetry in these parameterizations to provide alternative scaling rules [Blake et al., 2025], extended to meta learning [Thérien et al., 2024], and applied to newer optimizers [Essential AI et al., 2025]. Generally, in practice, a network where the learning rate is adequately tuned on a small-width network can be scaled using μP to be used as the learning rate for a larger width (theoretically, ∞-width), and still exhibit stable training dynamics (no blowup of L1 norms of pre-activations or output logits). While μP does not inherently predict loss values, its ability to preserve feature learning remains effective across diverse empirical settings [Vyas et al., 2023]. Consequently, hyperparameter transfer is well-aligned with the predictable loss trajectories observed in language model training [Dey et al., 2024, Bjorck et al., 2025].

Scaling laws formalize these trajectories by deriving parametric power-law fits of optimal loss against compute, dataset size, or parameter count [Kaplan et al., 2020, Hoffmann et al., 2022, Sorscher et al., 2022, Alabdulmohsin et al., 2023, Caballero et al., 2023]. Given a pretraining dataset and model architecture type, multiple models are trained varying the number of tokens seen, the model hyperparameters (including batch size), and the model sizes (width-depth). The resulting Pareto front characterizes the loss-compute trade-off, which is approximately linear on log-log axes and is commonly modeled with a power-law fit. This fit can then be used to predict the tokens required for compute-optimal loss given a fixed model, or the model size required for compute-optimal loss given a fixed token set to process. Chinchilla [Hoffmann et al., 2022] notably showed that this fit is sensitive to the hyperparameter choices, e.g., the length of cosine decay in the learning rate schedule. Subsequent works have looked at such a role of hyperparameters when scaling models [Porian et al., 2024], and also the suitability of certain hyperparameters to allow for compute reuse while scaling [Hägele et al., 2024]. Often, the role of *optimal* hyperparameters is highlighted as crucial but inadequately reported. Hyperparameter transfer methods become a key ingredient of a scaling recipe, and also influence novel observations for the scaling regime, such as scaling collapse [Qiu et al., 2025]. Bergsma et al. [2025] look at how μP, collapsed scaling curves, and novel parametric forms for scaling relationships together can be leveraged for early stopping learning curves.

B Synthetic Regression Benchmark

We use the synthetic regression benchmark in which the target function is constructed to have a power-law Fourier spectrum [Qiu et al., 2025]. Following their compute accounting, training progress is measured by the number of *processed training examples*; throughout, we refer to one regression example as one "token". To enable extensive hyperparameter optimization, we scale down the models while keeping the task and accounting unchanged. We fix the MLP depth to $L = 3$ hidden layers and scale width (n_{embd}), resulting in the model sizes in Table 1.

Data Generation. Each regression token consists of an input vector $x \in \mathbb{R}^d$ sampled from a fixed distribution (uniform on a bounded domain in the standard setup), and a scalar target

$$y = f^\star(x) + \epsilon, \tag{3}$$

where ϵ is i.i.d. noise and $f^\star$ is a random function whose Fourier coefficients decay as a power law. Concretely, $f^\star$ is constructed by sampling Fourier amplitudes with

$$\mathbb{E}\big[|\hat{f}(\omega)|^2\big] \propto \|\omega\|^{-\alpha}, \tag{4}$$

so that the target spectrum follows a controlled slope α.

[4] and the μP Github page: `https://github.com/microsoft/mup/blob/main/README.md`

Table 1: Synthetic MLP model scales used in the regression experiments. All models have $L = 3$ hidden layers; we scale only the hidden width n_{embd}. Parameter counts are rounded to one decimal place.

Width (n_{embd})	**Params (M)**
48	0.1
68	0.1
96	0.2
136	0.4
192	0.9
272	1.8
384	3.5

Model. We train width-scaled MLPs of depth $L = 3$. Let $h^{(0)} = x$ and for $\ell = 1, \ldots, L$,

$$h^{(\ell)} = \phi\Big(W^{(\ell)} h^{(\ell-1)} + b^{(\ell)}\Big), \qquad \hat{y} = W^{(L+1)} h^{(L)} + b^{(L+1)}. \tag{5}$$

We scale the hidden dimension n_{embd} while keeping depth fixed.

Although architecturally simple, these MLPs follow the parameterization and normalization conventions of Transformer feedforward blocks, with attention removed; thus they are not treated as generic fully-connected networks but as "Transformer MLP-only" models to preserve scaling-recipe behavior under the same compute accounting.

B.1 Training Horizon via *tokens/parameter*

For a model with P parameters, we set the training horizon using a *tokens/parameter* budget τ and process a total of

$$T \;=\; \tau \cdot P$$

training examples (tokens). Given batch size B, this corresponds to T/B optimizer steps. We run experiments for $\tau \in \{20, 35, 50\}$.

Qiu et al. [2025] measure compute in processed examples and, for some comparisons, estimate a compute-optimal horizon as a function of model size (via a loss–compute frontier under a reference schedule) and then reuse that horizon when comparing schedules. In our setting, we fix τ explicitly (rather than re-estimating compute-optimal horizons) so that large, matched hyperparameter sweeps remain feasible and comparable across scales.

B.2 Hyperparameter Search Spaces

We use a deterministic Cartesian-product grid search with a fixed seed (no adaptive tuning). To keep the overall cost manageable, we distinguish: (1) a *base grid* evaluated at the smallest width, used to identify strong configurations and for transfer experiments; and (2) a *main grid* used for the majority of runs across target widths.

Grid Definitions. Table 2 lists both grids. The base grid expands the learning-rate and batch-size options, while the main grid keeps a smaller but representative range. In all cases the reported distributions (e.g., Fig. 2) are over the *full grid at the target width*.

Grid Sizes. The base grid contains $8 \cdot 3 \cdot 5 \cdot 3 \cdot 2 = 720$ configurations. The main grid contains $4 \cdot 3 \cdot 3 \cdot 3 \cdot 2 = 216$ configurations. Unless stated otherwise, each experimental setting uses the full main grid (216 evaluations).

B.3 Hyperparameter Importance

We analyze hyperparameter importance using the results of our deterministic grid searches: the reduced grid is evaluated at target scales, while the full grid is evaluated at the base scale (cf. Table 2). For each experimental setting, we treat the best validation loss achieved during training (best-so-far

Table 2: Hyperparameter search spaces for the regression benchmark. The base grid is evaluated at the smallest width (48) and used for transfer; the main grid is used for the primary sweeps across target widths.

Hyperparameter	Values
Base grid (evaluated at width = 48)	
Learning rate	{6e-4, 8e-4, 1e-3, 1.4e-3, 1.8e-3, 3e-3, 4e-3, 5e-3}
Weight decay	{0, 1e-5, 1e-2}
Batch size	{256, 512, 1024, 2048, 4096}
Warmup fraction	{0.01, 0.03, 0.05}
Cooldown fraction	{0, 0.2}
Main grid (used for target widths)	
Learning rate	{6e-4, 1e-3, 3e-3, 4e-3}
Weight decay	{0, 1e-5, 1e-2}
Batch size	{256, 512, 1024}
Warmup fraction	{0.01, 0.03, 0.05}
Cooldown fraction	{0, 0.2}

over processed examples) as the objective and compute hyperparameter importances using Optuna's fANOVA [Hooker, 2007] implementation.

Across all settings, learning rate is consistently the dominant contributor to performance variation, followed by the learning-rate schedule shape (primarily cooldown fraction, and to a lesser extent warmup fraction). In contrast, weight decay receives near-zero importance in all experiments. We attribute this to the small-model regime in our scaled-down benchmark (Table 1), where explicit ℓ_2 regularization has little effect compared to optimization and schedule choices. Consequently, while we keep weight decay in the search space for completeness and comparability, we do not emphasize it when interpreting transfer behavior and sensitivity analyses.

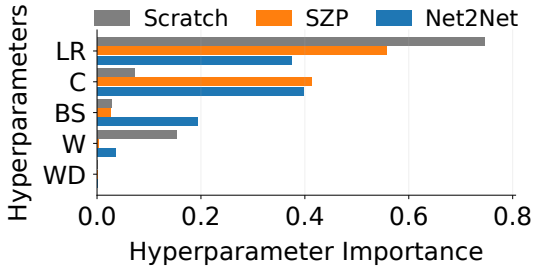


Figure 6: Hyperparameter importance (fANOVA via Optuna), aggregated across all target widths, computed on the main-grid search space (Table 2). We report importances separately for `Scratch`, `SZP` warmstarting, and `Net2Net` warmstarting. Abbreviations: LR = learning rate, WD = weight decay, BS = batch size, W = warmup fraction, C = cooldown fraction.

B.4 Details on μParameterization

Changing the width of a neural network changes not only its parameter count, but also the effective scale of activations, gradients, logits, and parameter updates. Therefore, hyperparameters tuned at one width need not transfer reliably to another. This creates a practical confounder for `WS` experiments: if each target width is retuned independently, improvements may come from better target-scale hyperparameters rather than from the growth operator itself.

μParameterization [Yang et al., 2021] addresses this issue by prescribing width-dependent initialization and learning-rate scalings. Its goal is to keep models of different widths in a comparable feature-learning regime, so that the effect of parameter updates on activations remains stable as width changes. Equivalently, μP can be viewed as a hyperparameter-transfer rule from a small base model to a larger target model.

Table 3: Compact summary of the μP transfer rules used in our experiments. Here, n denotes the width-like dimension for tensors whose fan-in/fan-out grows with model width. For attention logits, the scaling uses the per-head dimension d_{head}, which is fixed to 64 in our language-model width-scaling setup.

Group	**Forward / init**	**Adam LR**
Input / vector-like	$\text{Var}(W) \propto 1/n$	constant
Hidden / matrix-like	$\text{Var}(W) \propto 1/n$	$\propto 1/n$
Output / readout	multiplier $\propto 1/n$	constant
Attention logits	$q^\top k/d_{\text{head}}$	–
Scalar-like	constant	constant

Given a base-width configuration

$$h_{n_0} = (\eta, \lambda_{\text{wd}}, B, w_{\text{warmup}}, w_{\text{cooldown}}, \ldots),$$

we obtain the target-width configuration by

$$h_n = T_{\mu\text{P}}(h_{n_0}; n_0, n),$$

where $T_{\mu\text{P}}$ applies parameter-type-specific scaling rules to initialization and optimizer hyperparameters. The main rules are summarized in Table 3. The key point is that different tensors can have different width dependencies: input weights, hidden weights, readout weights, embeddings, normalization parameters, and attention-logit scaling are not all treated in the same way. In practice, we specify base shapes for the model, identify which dimensions grow with width, and apply the corresponding μP optimizer rules.

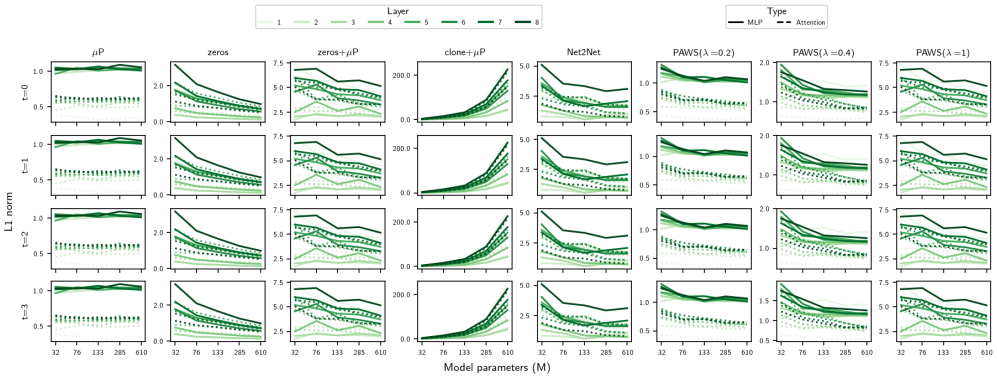


Figure 7: L1 norm of the layer activations across model sizes, for μP, `SZP`, `Net2Net` and the other naive baselines discused in Section 3, for $t = \{1, \ldots, 4\}$. The expectation is for each line to resemble the μP behaviour. However, in practice, and in expectation of the theory, the norms *not* blowing up with increasing width is adequate for stable hyperparameter transfer. We find that for most cases at larger widths, the activations become more stable and constant in width, especially for `SZP` (with $\lambda_{\text{shrink}} = 0.2$). Only for `Clone` as g do we observe the prohibitive instability with increasing width.

We use μP only as a practical mechanism for hyperparameter transfer. In the synthetic MLP benchmark, we can compare it directly against dense target-width tuning and against *static transfer*, where the best base-width hyperparameters are reused unchanged. In Figure 2, stars denote μP transfer and open circles denote static transfer. For language modeling, full target-scale sweeps are too expensive, so we tune at the smallest relevant scale and transfer with μP throughout. In Figure 7, we verify the baselines and algorithms used in the empirical experiments in this work based on the scaling of the activations with increasing width as recommended by Yang et al. [2021].

B.5 Hyperparameter Transfer across Widths

We study whether hyperparameters selected at the smallest width transfer to larger widths. For each target scale, we evaluate: (1) the full main grid sweep at the target width, and (2) a single transferred

run obtained by mapping the best configuration found at the base scale (width = 48) to the target width.

Transfer Rules. Under μP, the learning rate selected at the base width is transferred according to the μP prescription (i.e., it can be reused across widths when training uses the μP parameterization). In addition, following subsequent work on stabilizing width transfer, we scale batch size with target width as

$$B_{\text{target}} \;\approx\; B_{\text{base}} \cdot \sqrt{\frac{n_{\text{embd@target}}}{n_{\text{embd@base}}}},$$

where n_{embd} denotes model width.

C Design Intuition and Ablations for SZP

This appendix complements the empirical setup in Section 4.1 by analyzing the main design choices in SZP. The method combines three simple ingredients: zero-padding, perturbation, and shrinkage. Zero-padding embeds the trained base model into the larger model, perturbation activates the newly added parameters, and shrinkage reduces the dominance of the copied base-model path. We first give a simple theoretical intuition for these roles, then provide a mechanistic interpretability analysis of their effect on weight statistics during training, and finally ablate the two nontrivial design choices: the perturbation scale and the shrinkage factor.

C.1 Theoretical Intuition for SZP

Zero-padding as an Exact Embedding. Consider a feed-forward network with hidden states

$$h_{\ell+1} = \phi(W_\ell h_\ell + b_\ell),$$

where $\phi(0) = 0$. Let $P(\theta)$ denote the width expansion from width n to width $m > n$ by adding new neurons and setting every parameter touching them to zero. If we embed each hidden state as

$$\tilde{h}_\ell = \begin{bmatrix} h_\ell \\ 0 \end{bmatrix},$$

then the widened layer acts as

$$\tilde{h}_{\ell+1} = \phi\left(\begin{bmatrix} W_\ell & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} h_\ell \\ 0 \end{bmatrix} + \begin{bmatrix} b_\ell \\ 0 \end{bmatrix} \right) = \begin{bmatrix} \phi(W_\ell h_\ell + b_\ell) \\ 0 \end{bmatrix} = \begin{bmatrix} h_{\ell+1} \\ 0 \end{bmatrix}.$$

By induction over layers,

$$F_{P(\theta)}(x) = f_\theta(x) \qquad \text{for all } x.$$

Thus, in this idealized unnormalized setting, zero-padding preserves the pretrained function exactly. For normalized architectures such as Transformers with LayerNorm or RMSNorm, exact function preservation may additionally require coordinated rescaling of normalization parameters or adjacent weights – without this, zero-padding should be viewed as an embedding intuition rather than a strict function-preserving operation.

However, this exact embedding is also restrictive. Let

$$\mathcal{M} := \{P(\theta) : \theta \in \Theta_{\text{small}}\}$$

be the embedded small-model manifold. At points in $\mathcal{M}$, each newly added neuron has zero activation and zero outgoing weight. By the chain rule of backpropagation, because the outgoing weights are exactly zero, the error signal propagated back to these new neurons is zero. Consequently, the gradients with respect to all newly added parameters vanish, and under standard gradient descent,

$$\nabla_{\Theta_{\text{new}}} \mathcal{L}(P(\theta)) = 0.$$

Thus, pure zero-padding can keep optimization on the embedded small-model manifold, preventing the wider model from using its additional capacity.

Why Perturbation and Shrinkage Help. The SZP initialization can be written as

$$\Theta_0 = \lambda_{\text{shrink}} P(\theta^\star) + E, \qquad 0 < \lambda_{\text{shrink}} \leq 1,$$

where $\theta^\star$ is the pretrained base-model solution and E is a perturbation.

For one widened layer, partition old and new coordinates and write the input as

$$\tilde{h} = \begin{bmatrix} h \\ h_{\text{new}} \end{bmatrix},$$

where $h_{\text{new}} = 0$ for pure zero-padding and $h_{\text{new}} = O(\|E\|)$ after perturbations from preceding layers. The following calculation keeps only first-order terms in the perturbation magnitude. Let the perturbation matrix and bias vector be conformably partitioned into blocks corresponding to the old (o) and new (n) dimensions:

$$E = \begin{bmatrix} E_{oo} & E_{on} \\ E_{no} & E_{nn} \end{bmatrix}, \qquad e = \begin{bmatrix} e_o \\ e_n \end{bmatrix}.$$

The preactivation then takes the form

$$\tilde{z} = \begin{bmatrix} \lambda_{\text{shrink}}(Wh + b) \\ 0 \end{bmatrix} + \begin{bmatrix} E_{oo}h + e_o \\ E_{no}h + e_n \end{bmatrix} + O(\|E\|^2).$$

The lower block,

$$E_{no}h + e_n,$$

is the first nonzero signal available to the newly added neurons. If $E = 0$, the new coordinates remain inactive. Thus, perturbation moves the model off the embedded manifold $\mathcal{M}$ and activates the added degrees of freedom.

Shrinkage plays a complementary role. It reduces the copied old-path signal from $\|Wh + b\|$ to approximately $\lambda_{\text{shrink}}\|Wh + b\|$, while the new-path signal remains controlled by the perturbation.

The relative influence of the new coordinates is therefore, to first order,

$$\frac{\|(\tilde{z})_{\text{new}}\|}{\|(\tilde{z})_{\text{old}}\|} = \frac{\|E_{no}h + e_n + O(\|E\|^2)\|}{\|\lambda_{\text{shrink}}(Wh + b) + E_{oo}h + e_o + O(\|E\|^2)\|}.$$

When the copied old-path signal dominates the old-block perturbation, i.e.,

$$\lambda_{\text{shrink}}\|Wh + b\| \gg \|E_{oo}h + e_o\|,$$

this simplifies to

$$\frac{\|(\tilde{z})_{\text{new}}\|}{\|(\tilde{z})_{\text{old}}\|} \approx \frac{\|E_{no}h + e_n\|}{\lambda_{\text{shrink}}\|Wh + b\|}.$$

Taking $\lambda_{\text{shrink}} < 1$ makes it easier for the newly added neurons to compete with the copied subnetwork. Crucially, a larger relative preactivation ensures that the newly added weights generate correspondingly larger gradients during backpropagation, actively pulling the optimization trajectory away from the restrictive manifold $\mathcal{M}$. In summary, zero-padding preserves the pretrained model, perturbation activates the new dimensions, and shrinkage accelerates adaptation by breaking the optimization dominance of the inherited narrow solution.

C.2 Interpretability

In Figure 8 (left), we trace weight statistics for SZP vs. Zeros+Perturb (SZP without shrinking) across the input-output layers and the middle layer's attention and MLP matrices. The CL literature highlights shrinking's role in *preserving* the learned function via weight-norm clamping, which we observe here and correlate with improved plasticity [Lyle et al., 2023, Clare, 2025] and loss reduction (Figure 1). As a more informative proxy than L_1 or L_2, we track the $RMS \to L_\infty$ operator norm — the maximum row-wise L_2 norm scaled by the square root of the input width — which is crucial for transferring learning dynamics across scales [Filatov et al., 2025, Pethick et al., 2025]. Shrinking also improves utilization of new neurons, evidenced by the larger effective rank of the attention layer, and enables a larger shift from initialization ($||W_t - W_0||_2$), letting the grown model move further from its base weights. This matters because zero-padding or cloning alone may not break sufficient symmetry between old and new neurons [Yu et al., 2026], leaving the grown model too close to the base in function space, which manifests as loss saturation in Figure 1.

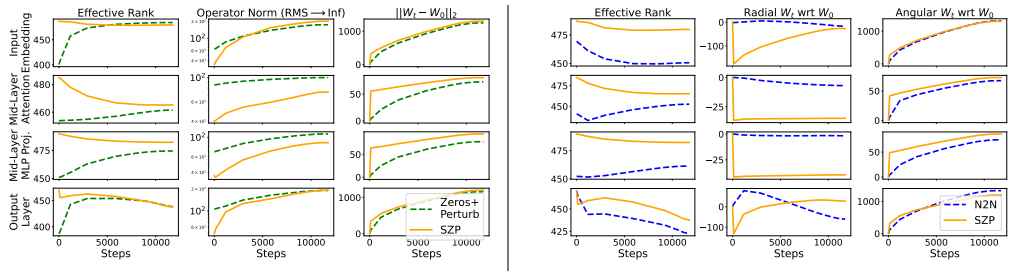


Figure 8: Looking inside WS for $14M \rightarrow 77M$: Plotting metric traces over steps; *Top row*: Input Embedding; *Middle rows*: The middle layer's attention and MLP matrices; *Bottom row*: Output Un-embedding; (*left*) Comparing `Zeros+Perturb` with *Shrink+Zeros+Perturb*(SZP), showing the *Effective Rank*, *L1 Norm*, and the norm of the difference of the weights at step t to the grown weight at initialization; (*right*) Comparing `Net2Net` with SZP, showing the *Effective Rank*, the norm of the weight at step t projected along the unit vector at initialization, and the norm of the weight at step t orthogonal to the unit vector at initialization. Refer to Section 4.1 for experiment details.

Table 4: Perturbation-scale ablation for SZP on a $32\text{M} \rightarrow 286\text{M}$ transfer. The table reports final validation loss; the corresponding learning curves are shown in Figure 9.

Hyperparameter	**Final validation loss**
Perturbation scale σ_{perturb}	
0	2.76
10^{-4}	2.50
10^{-3}	2.50
10^{-2}	2.49
10^{-1}	2.46
SZP $(1/\sqrt{\text{width}})$	**2.46**

C.3 Perturbation Scale Ablation

Perturbation is used to activate the newly added neurons after zero-padding. To study its sensitivity, we sweep the perturbation scale σ_{perturb} on a $32\text{M} \rightarrow 286\text{M}$ transfer, keeping the remaining SZP settings fixed. The default setting uses

$$\sigma_{\text{perturb}} = \frac{1}{\sqrt{\text{width}}}.$$

Table 4 reports the final validation loss for each perturbation scale, while Figure 9 shows the corresponding learning curves. Without perturbation, performance is substantially worse, suggesting that pure zero-padding does not effectively activate the added capacity. Nonzero perturbations improve both convergence and final loss, with the default $\sigma_{\text{perturb}} = 1/\sqrt{\text{width}}$ matching the best final performance in this sweep. The learning curves further show that the default reaches the low-loss region earlier than most fixed perturbation scales, which supports using it as a simple scale-aware choice rather than tuning σ_{perturb} separately for each transfer.

C.4 Shrinkage Ablation

Shrinkage controls the strength of the copied base-model path in SZP. In the initialization $\Theta_0 = \lambda_{\text{shrink}} P(\theta^\star) + E$, smaller values of λ_{shrink} reduce the contribution of the inherited narrow solution, while the perturbation E activates the newly added coordinates. As discussed in Section C.1, this makes it easier for the new neurons to compete with the copied subnetwork after growth.

This role of shrinkage is consistent with recent work on plasticity in warm-started networks. DASH [Shin et al., 2024] also uses shrinkage to mitigate plasticity loss, but does so in a direction-aware manner: weights associated with useful learned features are preserved more strongly, while components attributed to memorized noise are shrunk more aggressively. In contrast, SZP intentionally uses a single global shrinkage factor, making it architecture-agnostic and easy to combine with

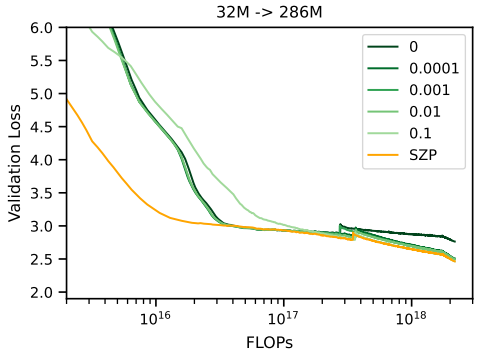


Figure 9: Effect of perturbation scale σ_{perturb} on a $32\text{M} \rightarrow 286\text{M}$ transfer. Validation-loss trajectories are shown as a function of training FLOPs for different fixed perturbation scales. The SZP default, $\sigma_{\text{perturb}} = 1/\sqrt{\text{width}}$, is highlighted in orange. Nonzero perturbations substantially improve over pure zero-padding, and the default reaches the low-loss regime quickly while matching the best final validation loss in the sweep.

arbitrary growth operators. Relatedly, weight clipping has been proposed as a simple way to control weight magnitudes and improve plasticity in continual and reinforcement learning settings [Elsayed et al., 2024]. Our use of shrinkage can be viewed as a one-shot initialization-time analogue of this broader norm-control principle.

We ablate λ_{shrink} while keeping the remaining SZP settings fixed. Figure 10 reports final validation loss across three base checkpoints, three token budgets, and multiple target scales. Across settings, the best values are concentrated in the low-to-moderate shrinkage regime, typically around $\lambda_{\text{shrink}} \in \{0.2, 0.4\}$. Setting λ_{shrink} too close to 1 corresponds to little or no shrinkage, leaving the grown model more dominated by the inherited base-model path. Conversely, setting it too close to 0 removes too much of the pretrained signal and makes the initialization closer to perturbation-only growth. The broad minimum around 0.2–0.4 suggests that shrinkage is useful, but does not require precise tuning.

We use $\lambda_{\text{shrink}} = 0.4$ in the main experiments as a stable default in this low-loss region. This choice is also consistent with the post-hoc analysis in Figure 8: compared with perturbation alone, adding shrinkage improves effective rank and permits larger movement from initialization, indicating that the grown model better uses its added capacity rather than remaining anchored to the base model.

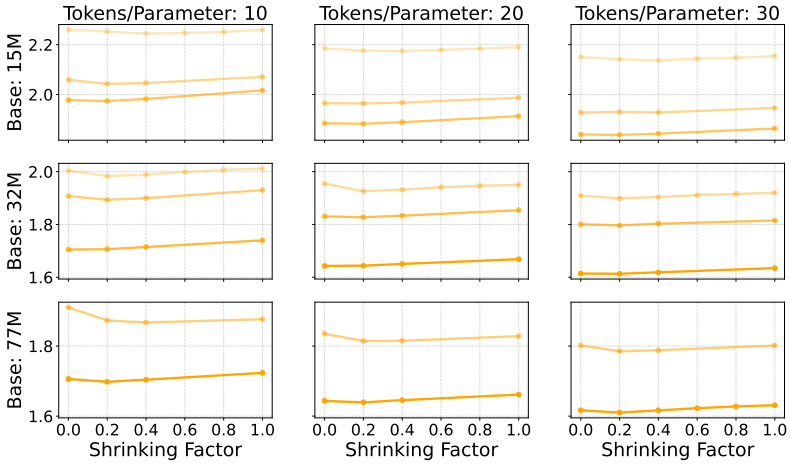


Figure 10: Ablation for shrinking factor (λ_{shrink}) for SZP. The rows represent a base checkpoint. Each line in the plot represents a target model scale warmstarted from the given checkpoint for the row. The darker the line, the larger the target size (larger growth factor).

Table 5: Language-model scales used in our experiments. All models have 8 layers and a sequence length 1024. We scale the width while keeping the attention head size fixed at 64. Parameter counts are rounded to the nearest million.

d_{model}	n_{head}	**Head size**	**Params (M)**
128	2	64	14
256	4	64	32
512	8	64	77
768	12	64	134
1280	20	64	286
2048	32	64	610
3072	48	64	1200

D Language Modeling Task Details

D.1 Model Scales

Table 5 summarizes the language-model scales used across our experiments.

In this work, we focus only on width-scaling, specifically the setting where the *head size*, i.e., the output dimension of each attention head, remains fixed across scales. Equivalently, this form of width-scaling adds *more heads* of the same size to each layer. This approach is standard in the literature, especially when studying feature learning under width-scaling [Everett et al., 2024]. We note that this form of width-scaling provides the fairest setup for comparing `WS` methods. The *query*, *key*, and *value* weights are grouped per head within a tensor.

For a simple mental illustration, a 1-head network with width 1 will have an attention tensor with 3 dimensions, one each for *query*, *key*, and *value*, assuming no weight sharing. When grown to width 2 by scaling the head dimension, the flattened attention tensor has 6 dimensions, 2 each for *query*, *key*, and *value*. This introduces a subtle layout inefficiency: under index-wise growth, the second dimension of the *query* block in the larger model corresponds to the *key* dimension of the smaller model. This correspondence is maintained cleanly when width-scaling is implemented by increasing only the number of heads while keeping the head size fixed.

D.2 Optimization Details

Optimizer. We optimize the cross–entropy loss with AdamW [Loshchilov and Hutter, 2019]. Unless otherwise stated, we keep the original moments $(\beta_1, \beta_2) = (0.9, 0.95)$ and numerical stabilizer $\epsilon = 10^{-8}$. Gradient global norm clipping is set to 1.0.

Adam ϵ. We set it to the PyTorch default of 10^{-8} that also follows the suggestion from Everett et al. [2024] for sub-billion parameter models.

Weight Decay. All experiments use *zero* weight decay. This isolates the effect of warmstarting from explicit ℓ_2 regularization, avoiding confounding interactions between the two mechanisms.

Warmup–Stable–Decay (WSD) Schedule. The learning rate follows a trapezoidal profile - let T be the total number of optimizer steps. For $\eta_{\max}$ the peak learning rate, warm-up fraction $w = 0.01$ and decay fraction $d = 0.20$, the instantaneous rate η_t is

$$\eta_t = \begin{cases} \eta_{\max} \frac{t}{wT}, & 0 \le t < wT, \\ \eta_{\max}, & wT \le t < (1-d)T, \\ \eta_{\max} \frac{T-t}{dT}, & (1-d)T \le t \le T. \end{cases}$$

Warm-up ensures numerical stability in the early regime [Kosson et al., 2024], while the constant plateau facilitates long, steady optimization at a single effective step size. The linear cool-down to zero replaces the abrupt drop used by step schedules and was found to improve final performance by Hägele et al. [2024].

Table 6: Base-scale language-model hyperparameters selected for μP transfer.

Params (M)	Selected $\eta_{\max}$	Effective batch size
14	3×10^{-3}	64
32	5×10^{-3}	128
77	2.5×10^{-3}	256

Training Budget. Unless otherwise stated, language-model runs use budgets of 10, 20, or 30 tokens per model parameter, with 20 *tokens/parameter* corresponding to the Chinchilla compute-optimal reference budget [Hoffmann et al., 2022]. For a given model scale, the number of optimizer steps is computed from the total token budget and the effective batch size in Table 6.

D.3 Grid Search Results for Optimal Hyperparameters

For the language-model experiments, we tune the peak learning rate $\eta_{\max}$ and effective batch size at small base scales, then transfer the selected configuration to larger target scales using the μP rules. This section only reports the base-scale choices used for transfer. The model sizes are listed in Table 5, and the shared optimizer and schedule settings are given in Section D.2.

For target-scale runs, per-device micro-batch sizes are chosen as the largest values that fit in memory on the available L40S GPUs, and gradient accumulation is used to realize the transferred effective batch sizes. The same base-scale selections are reused across the 10, 20, and 30 *tokens/parameter* budgets, with the number of optimizer steps adjusted according to the budget.

E Compute Resources

We report approximate compute for the experiments used in the paper in Tables 7 to 9. The synthetic MLP experiments were run on CPU nodes with dual-socket AMD EPYC 9655 processors, providing 192 physical cores in total. The language-model experiments were run primarily on NVIDIA L40S GPUs with 48GB VRAM using CUDA 12.7, with the 1.2B runs executed on H200 GPUs.

For the synthetic MLP experiments, Table 8 reports the 20 *tokens/parameter* runs, which include the base grid, main target-width grid, and transfer runs described in Appendix B.2. Additional 30 and 50 *tokens/parameter* MLP runs are transfer-only and are used for the IsoFLOP fits. For each budget, this gives 378 runs. These additional runs are summarized in Table 9. Together, the reported synthetic MLP experiments account for approximately 10,143 CPU-hours, or 423 CPU-days.

For the reported language-model experiments, we report GPU-hour ranges based on the number of completed runs, hardware allocation, and typical wall-clock time per target scale. Including an allowance for debug and re-runs, this corresponds to roughly 39,000–50,000 GPU-hours.

These estimates aggregate the completed runs used in the reported figures and exclude exploratory, failed, or auxiliary runs not used in the paper. The associated emissions footprint is expected to be lower than implied by raw compute alone, as the hosting facility is powered primarily by renewable electricity.

F Additional Empirical Results

F.1 LLaMa Architecture Results

To test whether the behavior of SZP is specific to the GPT-2 style architecture used in the main language-model experiments, we repeat the comparison on a LLaMA-style decoder architecture [Touvron et al., 2023]. We keep the same training budget of 20 *tokens/parameter* and use the same base-scale hyperparameter selection protocol for both Scratch and SZP. The only difference between the two runs is the initialization: Scratch is trained from a randomly initialized target model, while SZP initializes the target model by warmstarting from the corresponding base checkpoint. As shown in Figure 11, SZP consistently improves early convergence across all tested target sizes, suggesting that the gains are not tied to a particular decoder implementation.

Table 7: Approximate GPU compute used by target model scale for the reported language-model experiments. Unless otherwise noted, runs use NVIDIA L40S GPUs; the 1.2B runs use H200 GPUs.

Target scale (M)	**# runs**	**GPU-days / run**	**Total GPU-hours**
32	25	$1.5 - 2.0$	$900 - 1200$
77	40	$2.0 - 3.0$	$1920 - 2880$
134	49	$4.0 - 6.5$	$4700 - 7650$
286	53	$12 - 16$	$15300 - 20400$
610	13	32	10000
1200 (H200)	2	$56 - 64$	$2700 - 3100$
Subtotal	182	–	$35600 - 45200$
+debug/re-runs($\sim 10\%$)	–	–	~ 450.0
Total	–	–	$39000 - 50000$

Table 8: Approximate CPU compute used by target width for the reported 20 *tokens/parameter* width-scaling synthetic MLP experiments. This includes the base grid, target-width grid, and transfer runs.

Width	**# base widths**	**# runs**	**CPU-hours / run**	**Total CPU-hours**
48	–	720	0.17	120.0
68	1	234	0.27	62.4
96	2	252	0.48	119.7
136	3	270	0.92	247.5
192	4	288	1.81	520.0
272	5	306	4.31	1317.5
384	6	108	10.00	1080.0
Total	–	2178	–	3467.1

F.2 Downstream Task Results

To complement the validation-loss comparison in Section 4.2, we evaluate the same 1.2B-parameter `Scratch` and `SZP` checkpoints on a compact suite of downstream multiple-choice benchmarks using LIGHTEVAL [Habib et al., 2023]. These evaluations are intended as a sanity check that the gains from warmstarting are not limited to language-model validation loss, but also translate to standard commonsense, science, and general knowledge tasks. We report accuracy on ARC-Challenge [Clark et al., 2018], HellaSwag [Zellers et al., 2019], MMLU [Hendrycks et al., 2021], OpenBookQA [Mihaylov et al., 2018], PIQA [Bisk et al., 2020], and WinoGrande [Sakaguchi et al., 2020] in Table 10. For the subset overlapping with Hugging Face Open LLM Leaderboard v1, we follow the corresponding fixed few-shot settings: ARC-Challenge 25-shot, HellaSwag 10-shot, MMLU 5-shot, and WinoGrande 5-shot [Hugging Face, 2024a,b]. We additionally include OpenBookQA and PIQA as standard LIGHTEVAL multiple-choice tasks in the 0-shot setting. Across

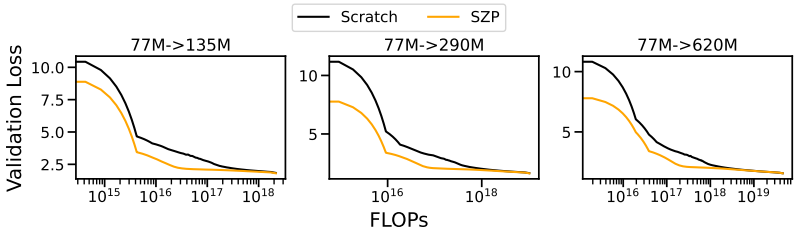


Figure 11: Comparing `SZP` with `Scratch` on a LLaMA-style decoder architecture [Touvron et al., 2023], trained for 20 *tokens/parameter*. Both methods use the same base-scale hyperparameter selection and transfer protocol; they differ only in initialization. `SZP` warmstarts the target model from the corresponding base checkpoint, while `Scratch` trains the target model from a random initialization.

Table 9: Additional CPU compute for transfer-only synthetic MLP runs used in the IsoFLOP fits.

Budget	# runs	Total CPU-hours
30 *tokens/parameter*	378	2503.4
50 *tokens/parameter*	378	4172.3
Total	756	6675.6

all six benchmarks, SZP improves over the corresponding Scratch baseline, suggesting that the optimization advantage observed in the pretraining curves carries over to downstream behavior.

Since these results are based on a single checkpoint per method and do not include multiple training seeds or statistical significance testing, we interpret them as a sanity check rather than a definitive downstream-performance claim.

Table 10: Downstream multiple-choice accuracy for the 1.2B-parameter checkpoints. Higher is better.

Method	ARC-C	HSwag	MMLU	OBQA	PIQA	WinoG.
Scratch	17.7	27.4	26.0	13.4	60.2	51.0
SZP	**20.1**	**29.4**	**26.4**	**15.8**	**64.6**	**53.0**

G Net2Net

Net2Net [Chen et al., 2016] is a seminal function-preserving growth method: it expands a trained *base* network into a larger *target* network by duplicating/splitting hidden units (width) and/or inserting identity layers (depth), such that the target initially computes (approximately) the same function as the base. In recent standardized evaluations of model growth for LLM pre-training, a Net2Net-style *direct* width-expansion operator is used as the canonical function-preserving width-growth baseline (often referred to as a "direct" growth operator) [Du et al., 2024]. We therefore adopt Net2Net as our primary width warmstarting baseline.

G.1 Why Net2Net?

We use Net2Net [Chen et al., 2016] because it (1) targets *width expansion*, which is the focus of this work, (2) is (approximately) function-preserving at initialization, and (3) requires no changes to the training objective or additional optimization stages. This makes it a clean and widely used baseline for isolating the effect of the warmstarted initialization. Moreover, in the large-scale growth benchmark of Du et al. [2024], the Net2Net-style *direct* widening operator is the strongest width-expansion baseline among the compared growth operators, motivating its use as our primary width warmstarting baseline.

Why not other Methods? We exclude several alternative strategies by design. Learned growth and distillation-based methods (for example, linear transformations [Wang et al., 2023a], progressive module training [Gong et al., 2019], or teacher–student frameworks [Qin et al., 2021]) introduce additional stages (e.g., auxiliary alignment losses or intermediate supervision), which makes it harder to attribute gains specifically to initialization quality. Importantly, our method could be composed with such techniques; however, such extensions are outside the scope of this work.

G.2 Net2Net Width Expansion

Intuition. Width growth in Net2Net can be viewed as *splitting* a hidden unit into several identical copies, then rescaling the outgoing weights so that the combined contribution of the copies matches the original unit. Figure 12 illustrates this matrix-level perspective: we first expand the input dimension (duplicate columns), then expand the output dimension (duplicate rows).

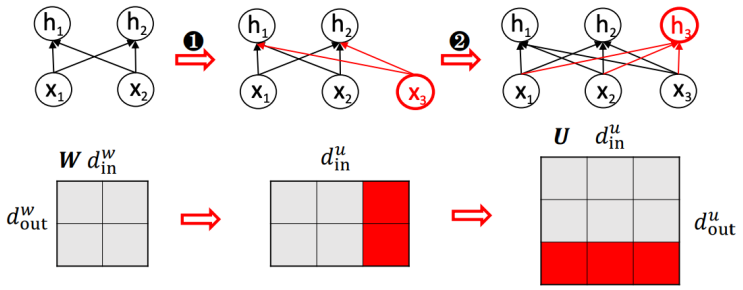


Figure 12: **Net2Net-style matrix expansion.** A width expansion can be implemented by duplicating columns (fan-in growth) and rows (fan-out growth), corresponding to adding new hidden units (highlighted). We use this figure (adapted from transformer growth descriptions building on `Net2Net` and `bert2BERT`) to illustrate the operation. Sourced from [Chen et al., 2021].

Neuron-splitting Formalism (single hidden layer). Consider a base MLP block with hidden width n,

$$h = \phi(W_{\text{in}}x), \qquad y = W_{\text{out}}h,$$

where $W_{\text{in}} \in \mathbb{R}^{n\times d}$ and $W_{\text{out}} \in \mathbb{R}^{d_{\text{out}}\times n}$. To grow the hidden width from n to q, `Net2Net` defines a mapping $\pi \in \{1,\dots,n\}^q$ that assigns each new unit j to a source unit $\pi(j)$. Let $c_i = |\{j : \pi(j) = i\}|$ be the copy count of base unit i. Then a function-preserving widening is achieved by,

$$(W'_{\text{in}})_{j,:} = (W_{\text{in}})_{\pi(j),:}, \tag{6}$$

$$(W'_{\text{out}})_{:,j} = \frac{1}{c_{\pi(j)}}(W_{\text{out}})_{:,\pi(j)}. \tag{7}$$

Immediately after expansion, all copies produce identical activations, and the rescaled outgoing weights ensure that the sum of their contributions equals the original unit, so $W'_{\text{out}}\phi(W'_{\text{in}}x) = W_{\text{out}}\phi(W_{\text{in}}x)$.

Matrix Form (general linear layers). The same idea can be written compactly at the matrix level. For a linear map $W \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$ expanded to $W' \in \mathbb{R}^{D_{\text{out}}\times D_{\text{in}}}$, Net2Net-style widening can be expressed as

$$W' = L\,W\,R, \tag{8}$$

where R duplicates (copies) input dimensions, i.e., columns of W, to reach D_{in}, and L duplicates output dimensions, i.e., rows of W, to reach D_{out}. To preserve the function, L and/or R additionally include a $1/c$ rescaling for duplicated units, where c is the copy count of the corresponding source unit. This is the same "copy-and-rescale" operator view commonly used when describing width growth in transformer blocks [Du et al., 2024].

Noise for Symmetry Breaking. Because duplicated units start identically, we optionally add small Gaussian noise to the newly created parameters (or to a subset of duplicated rows/columns) to break symmetry, following standard `Net2Net` practice [Chen et al., 2016]. In our experiments this noise is small enough to keep the initialization close to function-preserving while avoiding perfectly identical copies.

G.3 Implementation in our Experiments

MLPs. In the synthetic MLP experiments, we apply the widening rule above to each hidden layer: we duplicate hidden units according to π and rescale the corresponding outgoing weights by the copy counts, optionally injecting small Gaussian noise for symmetry breaking.

Transformers. In the transformer language-model experiments, we apply the same function-preserving principle when expanding width. Concretely, we grow matrices whose inner dimensions scale with model width (e.g., token embeddings and the projection matrices in attention/MLP blocks) via the same "copy columns / copy rows + rescale" recipe in Equation (8). For multi-head attention,

because we keep the head size fixed and grow width by adding heads, copying is performed at the level of complete heads: duplicated W_q, W_k, and W_v rows are left unscaled, while the copy-count rescaling is applied only to the corresponding W_o columns. This preserves the duplicated heads' softmax computation while averaging their combined contribution through the output projection. This yields an initialization that matches the base model computation as closely as possible at the moment of expansion, while remaining a drop-in replacement that does not modify the objective or training loop.

All comparisons use identical token budgets and optimization settings to isolate the effect of initialization.

G.4 Scope and Follow-up Work

While our study focuses on width-only growth, `Net2Net` [Chen et al., 2016] also proposes depth-expanding transformations (e.g., inserting identity layers). In transformers, `bert2BERT` [Chen et al., 2021] extends `Net2Net`-style ideas to BERT, and combines them with layer stacking to grow depth. More broadly, depth expansion can be paired with transformer-specific stacking strategies, such as those explored in *Stacking Your Transformers* [Du et al., 2024].

Our empirical study isolates *width scaling* (following Everett et al. [2024]) because our emphasis on hyperparameter transfer and scaling behavior requires extensive sweeps across widths, training horizons, and transfer settings. Nevertheless, extending our warmstarting analysis to depth is a natural next step, and should be compatible with depth-stacking mechanisms and depth-aware scaling rules (e.g., depth μP variants and recipe-based transfers such as completeP [Dey et al., 2025]). Ultimately, this points toward studying warmstarting under compound scaling; we first establish controlled evidence for hyperparameter transfer under width growth [Everett et al., 2024, Dey et al., 2025, Essential AI et al., 2025].

G.5 Comparison with `HyperCloning`

Table 11: Warmstarting from a *base→target* model size, trained for 20 *tokens/parameter*, comparing `HyperCloning` to `SZP` for up to 287M parameters. Both start from the same base checkpoints with μP-transferred hyperparameters and differ only in the growth operator. Validation loss at four matched FLOP levels per pair; the better value in bold. `SZP` reaches lower loss at every FLOP level, with the gap largest early in training.

32M → 77M			32M → 135M			77M → 287M		
FLOPs	`HyperCloning`	`SZP`	FLOPs	`HyperCloning`	`SZP`	FLOPs	`HyperCloning`	`SZP`
1.5e15	3.78	**3.26**	7.6e15	4.08	**3.01**	2.1e16	4.18	**3.14**
1.1e16	2.89	**2.32**	5.0e16	2.98	**2.24**	1.6e17	2.81	**2.02**
9.1e16	2.25	**2.19**	3.3e17	2.24	**2.08**	1.2e18	2.07	**1.87**
7.1e17	1.98	**1.93**	2.2e18	1.93	**1.83**	9.1e18	1.69	**1.67**

`Net2Net` is the canonical function-preserving width operator but predates Transformers. To check that our conclusions are not tied to it, we compare `SZP` against `HyperCloning` [Samragh et al., 2024], a recent function-preserving initialization for dense Transformers that tiles the base weights to fill the target width and rescales them so that the grown model reproduces the base model's outputs at initialization. Both methods start from the same base checkpoints, train for 20 *tokens/parameter*, and use the same μP-transferred hyperparameters (Section 4.1.2); the runs differ only in the growth operator.

Table 11 reports validation loss at four matched FLOP levels for three base→target pairs. `SZP` reaches lower loss at every FLOP level. The gap is largest early in training and narrows as training proceeds, as in the `Net2Net` comparison in Figure 3. We attribute the gap to the same symmetry argument as for `Net2Net` (Appendix C.2): cloned blocks start as exact duplicates, whereas `SZP` breaks symmetry at initialization. Exact function preservation additionally requires the target width to be an integer multiple of the base width; for the $512 \to 1280$ pair ($2.5\times$) we fill the remaining 256 columns with a half-tile of the base weights, so this run is function-preserving only *approximately*.

H Limits of Effective Warmstarting

H.1 Extended IsoFLOP Analysis

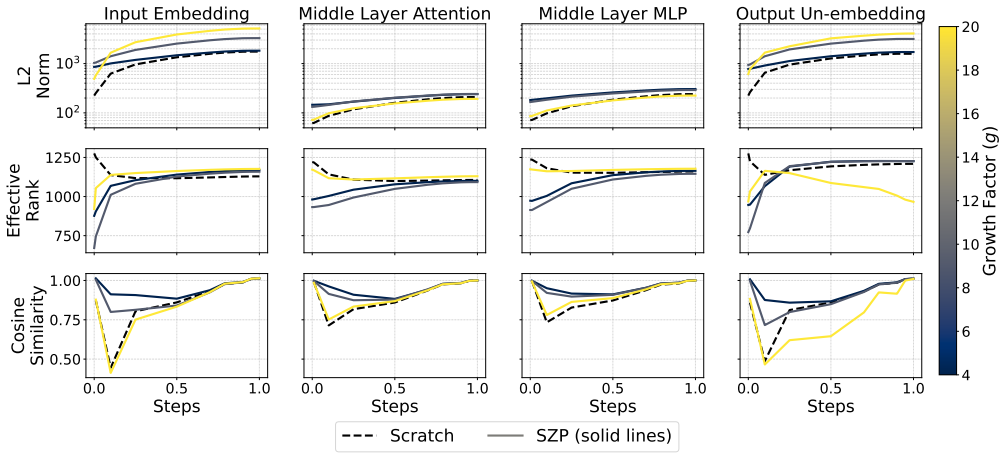


Figure 13: Mechanistic Interpretability for varying growth factors g for a 286M parameter LLM trained with 20 *tokens/parameter*, tracked across the input embedding, middle layer's attention and MLP matrices, and output unembedding. Solid lines show SZP at varying g (color scale), the dashed line shows training from scratch ($g = 1$). Cosine similarity is computed between consecutive checkpoints. At higher growth factors, `Warmstarting` increasingly resembles `Scratch`.

Mechanistic Interpretability of the Growth Factor. Figure 13 looks at the trace of weight statistics given increasing growth factor. The warmstarted models at higher g increasingly resemble `Scratch` across interpretability metrics, suggesting gradual dilution of the base model's structure and explaining the degradation in performance.

Extended Results. We extend the IsoFLOPs plots from Figure 4 by including more target scales and token budgets. Figures 14 and 15 show the results for `SZP` and `Net2Net` in the LLM setting, while Figures 16 and 17 show the corresponding results for the MLP setting. Similar to Figure 4, we can observe the existence of g_{upper} consistently across scales and budgets. For `Net2Net` in the MLP setting, the combination of strong performance and the limited range of growth factors at smaller target scales means g_{upper} occasionally falls outside the observed data and is omitted.

Including the Base Compute. Our primary analysis assumes that the base model is given and therefore excludes its training compute. Prior work has shown that the compute invested in pretraining the base model influences the effectiveness of subsequent model reuse [Liew and Kato, 2025]. We additionally investigate how accounting for the base-model training compute C_{base} affects our IsoFLOP analysis, as reported in Table 13. Including C_{base} shifts both g_{upper} and g_{opt} toward smaller growth factors, with both reaching $g = 1$ at the two largest budgets. Thus, while accounting for base-model compute changes the location of g_{upper} and can eliminate the advantage of warmstarting, our central observation remains unchanged: the range of growth factors for which warmstarting is beneficial is bounded.

Bootstrap Uncertainty. We use bootstrap resampling to quantify uncertainty in g_{opt} and g_{upper}. For each FLOPs budget, we resample the $g > 1$ points with replacement and refit the quadratic in $\log g$, keeping Scratch ($g = 1$) fixed as a common reference. Table 12 reports means and sample standard deviations from 1,000 valid bootstrap fits. We omit estimates where the full-data fit yields $g_{\text{opt}} > g_{\text{upper}}$. Uncertainty varies across methods, target scales, and compute budgets, without establishing uniformly greater precision for g_{upper}. While g_{opt} specifies a growth factor to target, g_{upper} marks the upper end of a beneficial range. This flexibility is a practical benefit of reporting g_{upper}, as practitioners can choose conservatively within this range without needing to match the boundary precisely.

Reliability of g_{opt}. Note that following Du et al. [2024], $g = 1$ corresponds to training from scratch rather than warmstarting from a same-sized model. The quadratic fit assumes a minimum, i.e. g_{opt}, but this minimum may be an artifact of this conceptual mismatch: warmstarting from a same-sized model could perform comparably to or better than warmstarting at slightly larger growth factors, and we do not sample densely enough at low growth factors to distinguish these cases. We therefore consider g_{upper} the more practically relevant quantity, while acknowledging that both estimates are subject to fitting uncertainty.

Toward a Unified Scaling Law. Our IsoFLOP analysis models the effect of the growth factor g on loss, while our scaling law fits capture the joint effect of model size N and training budget D. Since both can be modeled independently and predictably, a unified parametric form $L(N, D, g)$ that captures all three axes in a single scaling law is a natural next step. We consider this a promising direction for future work.

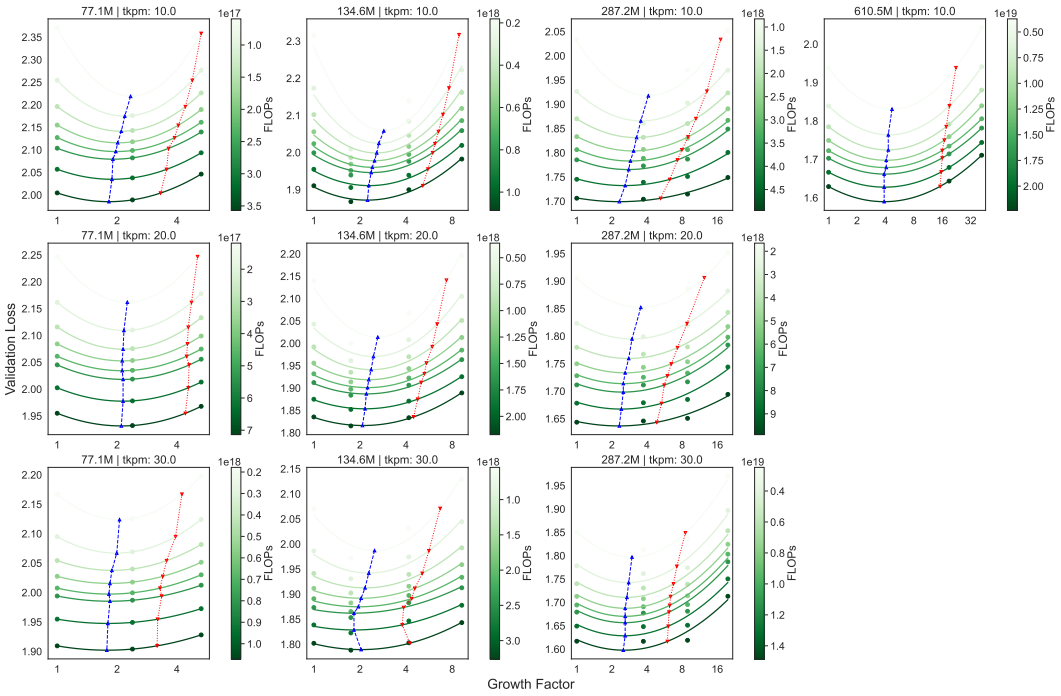


Figure 14: IsoFLOPs for SZP in the LLM setting. Blue and red lines denote g_{opt} and g_{upper} respectively, as in Figure 4.

H.2 Scaling Law Analysis

Fitted Parameters. Table 14 reports the fitted parameters and fit quality for the scaling laws shown in Figure 5. For reference, we repeat the parametric form. Following Approach 3 of Hoffmann et al. [2022], we model the loss as a function of the number of parameters N and training tokens D:

$$L(N, D) = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}},$$

where E, A, α, B, β are fit to the data. The resulting fits capture the data well, with all R^2 values exceeding 0.99.

Further Applications. So far, we have used scaling-law fits to characterize when warmstarting is effective compared with training from scratch. The same approach can also be used for comparing `Warmstarting` methods against each other. In Figure 18, we compare `SZP` and `Net2Net` by plotting isolines of the difference between their fitted loss surfaces across model sizes and training budgets. In the LLM setting, `SZP` is predicted to outperform `Net2Net` across all configurations. In the MLP setting, `Net2Net` is competitive at low token budgets, where its function-preservation properties

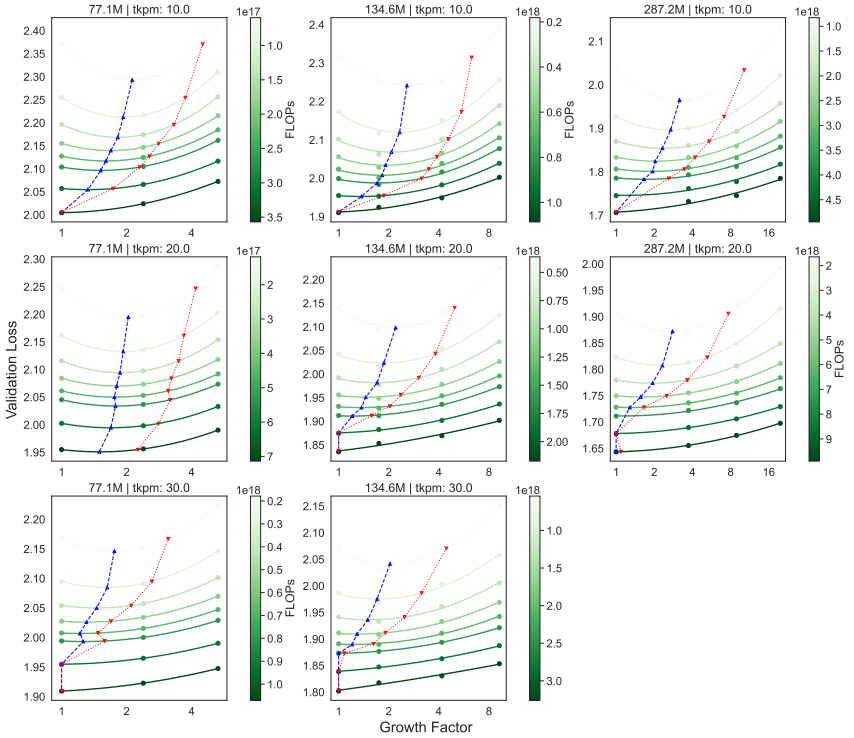


Figure 15: IsoFLOPs for `Net2Net` in the LLM setting. Blue and red lines denote g_{opt} and g_{upper} respectively, as in Figure 4.

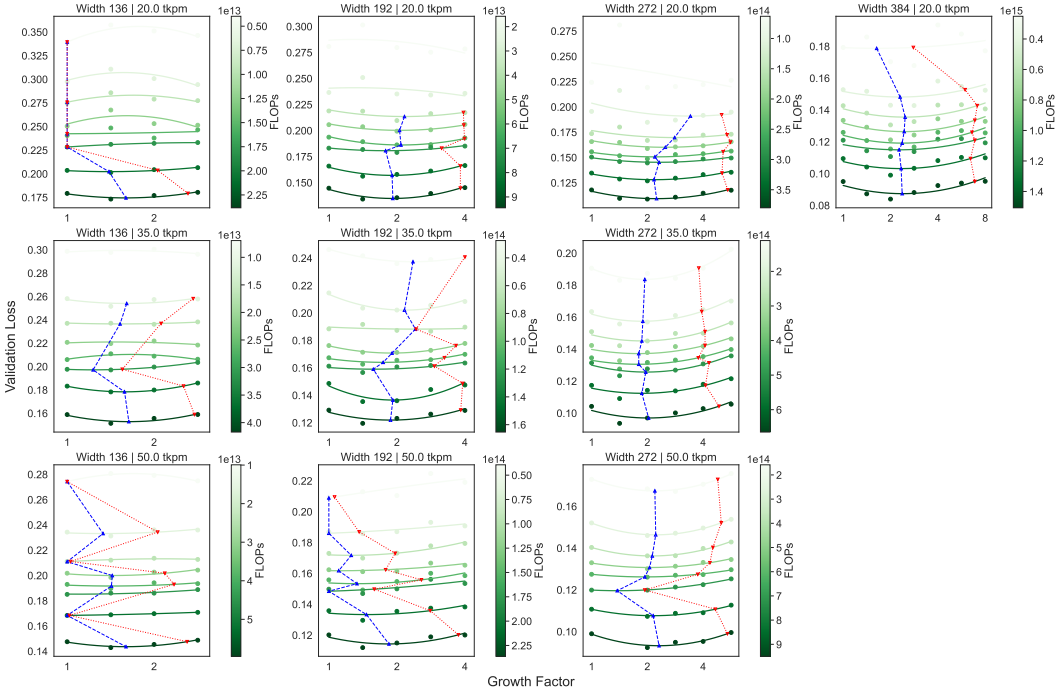


Figure 16: IsoFLOPs for SZP in the MLP setting. Blue and red lines denote g_{opt} and g_{upper} respectively, as in Figure 4.

Table 12: Bootstrap estimates (mean $\pm$ sample standard deviation) of g_{opt} and g_{upper} for SZP and Net2Net at 20 tokens/parameter, using 1,000 valid resampled quadratic IsoFLOP fits per budget with Scratch held fixed. Dashes indicate omitted estimates where the full-data quadratic opens downward or yields $g_{\text{opt}} > g_{\text{upper}}$.

	SZP		Net2Net	
FLOPs	g_{opt}	g_{upper}	g_{opt}	g_{upper}
LLM - Target 134.6M				
3.6×10^{17}	2.61 ± 0.19	6.94 ± 1.00	2.20 ± 0.08	4.92 ± 0.35
6.2×10^{17}	2.46 ± 0.15	6.17 ± 0.77	1.94 ± 0.11	3.81 ± 0.40
8.8×10^{17}	2.38 ± 0.13	5.74 ± 0.62	1.74 ± 0.10	3.07 ± 0.36
1.1×10^{18}	2.30 ± 0.11	5.34 ± 0.52	1.53 ± 0.11	2.40 ± 0.33
1.4×10^{18}	2.26 ± 0.09	5.18 ± 0.43	1.43 ± 0.11	2.10 ± 0.29
1.7×10^{18}	2.20 ± 0.08	4.92 ± 0.37	1.26 ± 0.11	1.63 ± 0.24
1.9×10^{18}	2.17 ± 0.06	4.71 ± 0.25	1.00 ± 0.00	1.00 ± 0.01
2.2×10^{18}	2.11 ± 0.05	4.49 ± 0.18	1.00 ± 0.00	1.00 ± 0.00
LLM - Target 287.2M				
1.6×10^{18}	3.48 ± 0.08	12.04 ± 0.53	2.74 ± 0.11	7.55 ± 0.56
2.8×10^{18}	3.02 ± 0.09	9.05 ± 0.50	2.31 ± 0.03	5.28 ± 0.14
4.0×10^{18}	2.78 ± 0.19	7.58 ± 1.07	1.91 ± 0.01	3.67 ± 0.04
5.2×10^{18}	2.60 ± 0.23	6.62 ± 1.25	1.58 ± 0.04	2.50 ± 0.13
6.4×10^{18}	2.46 ± 0.28	5.95 ± 1.42	1.30 ± 0.08	1.66 ± 0.21
7.5×10^{18}	2.29 ± 0.38	5.21 ± 1.74	–	–
8.7×10^{18}	2.19 ± 0.43	4.80 ± 1.88	1.00 ± 0.00	1.01 ± 0.01
9.9×10^{18}	2.00 ± 0.57	4.18 ± 2.21	1.00 ± 0.00	1.02 ± 0.02
MLP - Target width 272				
6.3×10^{13}	–	–	2.24 ± 0.16	4.94 ± 0.69
1.1×10^{14}	3.98 ± 1.48	3.26 ± 1.40	2.22 ± 0.17	4.78 ± 0.70
1.5×10^{14}	3.50 ± 1.36	3.79 ± 1.20	2.15 ± 0.16	4.80 ± 0.69
2.0×10^{14}	2.69 ± 0.71	5.30 ± 0.25	2.11 ± 0.14	4.78 ± 0.63
2.4×10^{14}	2.24 ± 0.44	4.96 ± 0.53	2.10 ± 0.13	4.76 ± 0.61
2.9×10^{14}	–	–	2.17 ± 0.11	4.93 ± 0.50
3.3×10^{14}	2.16 ± 0.10	4.87 ± 0.51	2.11 ± 0.14	4.79 ± 0.66
3.8×10^{14}	2.22 ± 0.13	5.08 ± 0.61	2.09 ± 0.13	4.67 ± 0.57
MLP - Target width 384				
2.5×10^{14}	2.58 ± 2.04	3.16 ± 1.99	2.51 ± 0.05	6.42 ± 0.33
4.3×10^{14}	2.26 ± 1.08	5.06 ± 1.81	2.44 ± 0.11	6.38 ± 0.57
6.1×10^{14}	2.46 ± 0.80	6.11 ± 1.52	2.47 ± 0.13	6.53 ± 0.75
7.9×10^{14}	2.45 ± 0.81	6.32 ± 1.39	2.41 ± 0.12	6.27 ± 0.63
9.7×10^{14}	2.34 ± 0.63	6.08 ± 1.33	2.40 ± 0.11	6.21 ± 0.59
1.1×10^{15}	2.33 ± 0.79	5.99 ± 1.42	2.48 ± 0.17	6.86 ± 0.95
1.3×10^{15}	2.33 ± 0.27	6.16 ± 1.05	2.38 ± 0.12	6.18 ± 0.64
1.5×10^{15}	2.40 ± 0.18	6.57 ± 1.04	2.34 ± 0.14	6.21 ± 0.73

are most beneficial, but SZP is predicted to perform better at larger token budgets. This aligns with the design of each method: Net2Net preserves the base model's function, providing an immediate advantage, while SZP enables the use of new neurons (see Section C.2), which becomes increasingly beneficial as the training budget grows and the model has more room to adapt. More broadly, this demonstrates that scaling law fits enable a richer comparison between warmstarting methods than evaluations at a single training budget, capturing how each method's effectiveness varies with both model size and training duration.

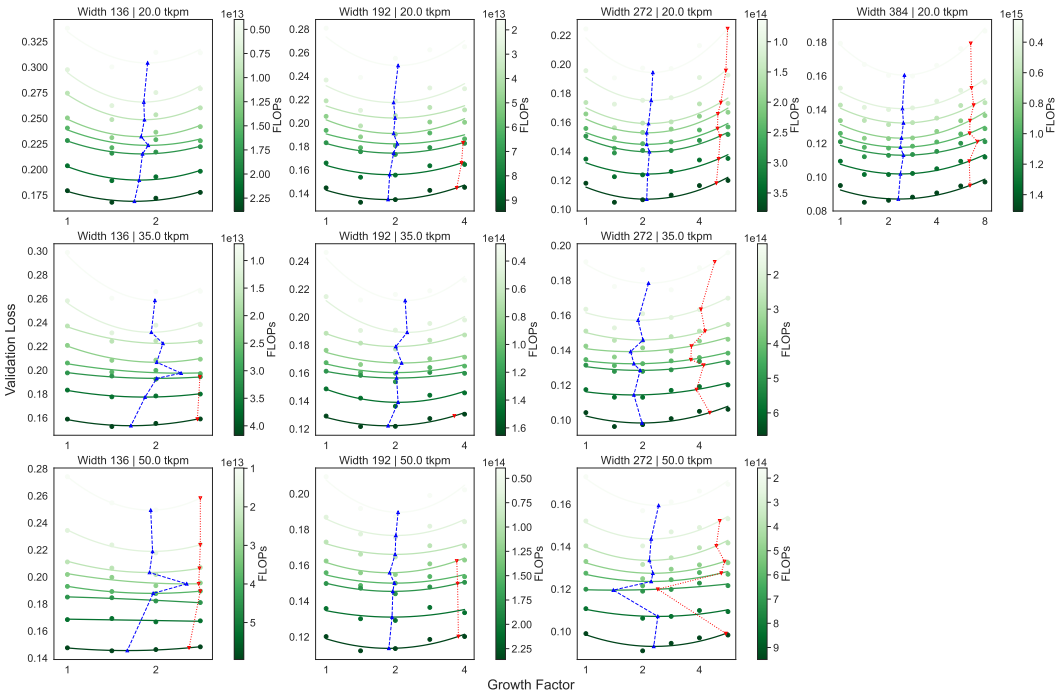


Figure 17: IsoFLOPs for `Net2Net` in the MLP setting. Blue and red lines denote g_{opt} and g_{upper} respectively, as in Figure 4. g_{upper} is omitted where it falls outside the range of collected data to avoid extrapolation.

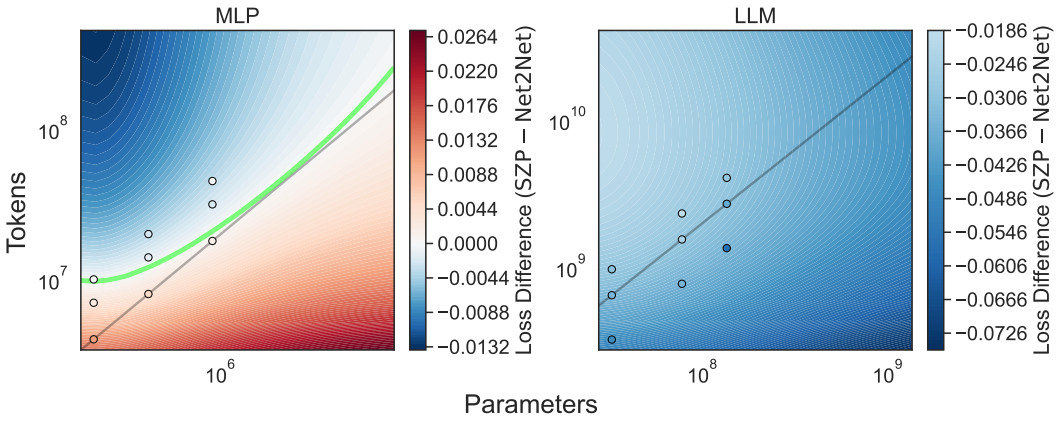


Figure 18: Isolines for *loss difference* comparing `SZP` and `Net2Net` for growth factor $g \approx 2$ (blue regions indicate `SZP` is better; red indicates `Net2Net` is better; the green line marks the boundary where both perform equally). In the LLM setting, `SZP` is predicted to outperform `Net2Net` across all configurations. In the MLP setting, `Net2Net` is competitive at lower token budgets, where its function-preservation properties are most beneficial, but `SZP` is predicted to perform better at larger token budgets.

Table 13: Effect of including the base-model training compute C_{base} in the IsoFLOP analysis, for SZP on the 287M LLM at 20 *tokens/parameter*. With C_{base} included, g_{upper} shifts to smaller growth factors and collapses to 1 at the largest budgets.

	g_{opt}		g_{upper}	
FLOPs	with C_{base}	without	with C_{base}	without
1.65×10^{18}	3.03	3.52	8.27	12.39
2.83×10^{18}	2.46	2.99	5.12	8.78
4.01×10^{18}	2.35	2.77	4.41	7.12
5.18×10^{18}	2.28	2.61	4.15	6.13
6.36×10^{18}	2.25	2.53	4.03	5.69
7.54×10^{18}	2.25	2.46	3.97	5.28
8.72×10^{18}	1.00	2.42	1.00	5.04
9.90×10^{18}	1.00	2.28	1.00	4.61

Table 14: The fitted parameters and R^2 as a quality-of-fit metric for the scaling law equations of Figure 5.

	LLM			MLP		
	SZP	Net2Net	Scratch	SZP	Net2Net	Scratch
A	1038	1739	18807	10484	424	1437
α	0.439	0.474	0.629	1.027	0.737	0.908
B	68	194	96	774	427	87
β	0.257	0.311	0.267	0.578	0.555	0.424
E	1.278	1.377	1.395	0.075	0.074	0.065
R^2	0.9999	0.9929	0.9922	0.9994	0.9929	0.9922

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The empirical results in Section 4 support the main claims, and Section 5 summarizes the contributions and scope.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed explicitly in the Section 5.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The work is primarily empirical. Section 3 gives the method reasoning, and Appendix, Section C.1 provides theoretical intuition rather than formal theoretical results.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

Justification: The codebase concerning the model training can be accessed via `https://github.com/Neeratyoy/warmstarting_exps` Section 4, Section B, and Section D provide complete details of all public resources and setups used.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Detailed in Section 4, Section B, and Section D.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Training and evaluation details are provided in Section 4; Section B, and Section D of the Appendix.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Due to the compute requirements, experiments are not repeated with error bars. The study uses controlled settings and compares relative performance.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The Appendix, Section E reports the compute resources, and the figures report the FLOPs for the corresponding runs.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses public/open-source resources and studies LLM pretraining efficiency. To the best of our knowledge, it does not violate the NeurIPS Code of Ethics.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: The work is methodological and targets LLM pretraining efficiency, with no direct societal impact beyond possible indirect compute savings.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The work proposes a training technique and does not release models or datasets intended for direct use.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Suitable license selected during submission.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: The paper does not introduce or release new assets.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing experiments or research with human subjects.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve study participants or human-subject research.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: Used primarily for grammar checks, improving visualization aesthetics, and code assistance.