Don't Drop Dropout: Optimizing Layer Sparsity for Efficient LLM Training and Inference

Mostafa Elhoushi[†], Alex Pretko[‡], Nolan Dey[†], Bin Claire Zhang[†], Gavia Gray[†], Gurpreet Gosal[†], Abdulrahman Mahmoud[‡], Shane Bergsma[†], Joel Hestness[†]
[†]Cerebras Systems, [‡]MBZUAI
m.elhoushi@ieee.org, joel@cerebras.net

Abstract. Layer dropout (a.k.a. stochastic depth) has been shown to enable faster training, higher accuracy, and robustness to zero-shot layer pruning in both language and vision transformers. However, as models and datasets have scaled, dropout—particularly layer dropout—has largely disappeared from large language models (LLMs) pre-training recipes. While some prior work has reported that dropout can degrade accuracy, no comprehensive study has quantified, let alone mitigated, this effect. In this study, we show that layer dropout *should* be used in state-of-the-art LLM training, establishing best practices and scaling analysis for both training and post-training benefits. Concretely, with optimal layer distribution, time schedule, and optimizer hyperparameters, we observe that *at the same training FLOPs layer dropout leads to lower loss.* For a given number of training steps, LLMs can achieve lower or similar validation loss while saving upto 25% of training FLOPs. Moreover, layer dropout enables significant post-training optimizations, such as early exit, intermediate-layer skipping, and self-speculative decoding, yielding up to 1.5× inference speedup with negligible accuracy loss. Across more than 2400 training experiments, spanning models from 271M to 8.2B parameters and datasets up to 160B tokens, we demonstrate that these findings extend reliably to large-scale training regimes. All pre-training experiments were run on Cerebras CS-3 systems.

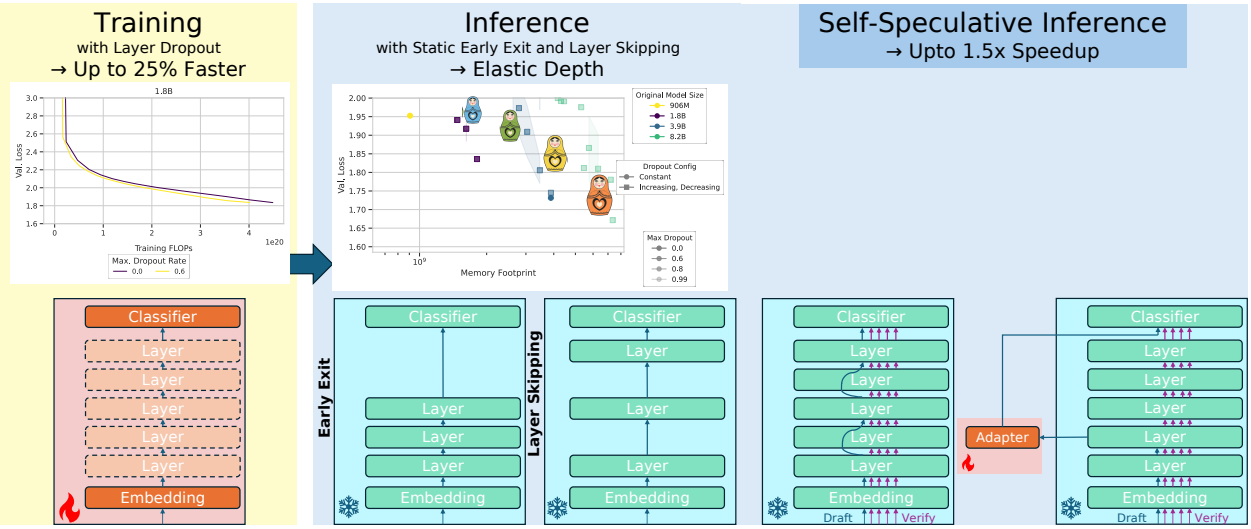


Figure 1: **Layer dropout as a unified mechanism for efficient LLM training and inference.** (*Left*) Layer dropout skips layers stochastically during pre-training, leading to faster training, and with our proposed configuration, does so without sacrificing validation loss. (*Center*) Trained models gain zero-shot "elastic depth," degrading gracefully under early exit and layer skipping. (*Right*) This robustness carries over to post-training adapters and self-speculative decoding for lossless inference speedup.

1 Introduction

Pretraining large language models (LLMs) demands extraordinary computational resources (Narayanan et al., 2021; Meng et al., 2025), where small improvements in time-to-accuracy can save millions of dollars (Coleman et al., 2019; Shen et al., 2024) and reduce carbon emissions (Acun et al., 2023; Wang et al., 2025).

Historically, regularization techniques improved validation accuracy for given training budgets by reducing overfitting and stabilizing optimization (Moradi et al., 2020; Wang & Manning, 2013; Murugan & Durairaj, 2017). *Dropout* was widely adopted in convolutional networks (Hinton et al., 2012) and early transformers (Vaswani et al., 2017). However, as LLMs scaled to billions of parameters and trillions of tokens, dropout has been largely abandoned (Raschka, 2025). Models trained for a single epoch over massive datasets have little opportunity for classical overfitting, and empirical evidence suggests activation dropout degrades performance under these conditions (Liu et al., 2025).

One type of dropout, Layer Dropout, also known as Stochastic Depth (Huang et al., 2016), can provide benefits beyond regularization. Unlike activation dropout or unstructured sparsity, which typically do not translate into wall-clock speedups due to sparse-kernel overheads, skipping entire transformer blocks yields structured sparsity that can reduce active training FLOPs almost linearly with the dropout rate (Zhang & He, 2020; Elkerdawy et al., 2021). It also encourages robustness to reduced-depth execution at inference time, enabling a single pretrained model to dynamically adapt to different latency and compute budgets without retraining. This supports zero-shot depth-wise optimizations such as elastic depth (Fan et al., 2020), early exit (Elhoushi et al., 2024), and intermediate layer skipping (Huang et al., 2016; Cai et al., 2021).

Despite layer dropout's promise, its role in state-of-the-art LLM pretraining has never been established through a comprehensive evaluation at scale. Existing evidence is fragmented across model families, dataset sizes, and implementation conventions. Many reported degradations may reflect suboptimal schedules or hyperparameters, rather than fundamental limitations. This leaves a basic unresolved question: *should layer dropout be used in modern large-scale LLM training, and if so, how should it be configured to preserve accuracy while delivering training and deployment benefits?*

We provide the first unified experimental study of layer dropout in LLMs, systematically varying (i) optimizer hyperparameters, (ii) depth-wise distribution and granularity of layer sparsity, and (iii) temporal dropout schedules, across fixed architecture and data. Across 2400+ training runs spanning 271M to 3.9B parameters and up to 116B tokens, we identify configurations that reliably improve training and inference efficiency. Our contributions are:

1. **Improved Compute–Accuracy Trade-offs:** Properly configured layer dropout reduces training FLOPs while achieving validation loss competitive with, and in several cases superior to, dense baselines at scale.
2. **Joint Optimization Framework:** We identify key interactions between dropout configurations, schedules, and optimizer hyperparameters that mitigate degradations observed in prior work.
3. **Depth-Elastic Inference:** The *average training dropout rate* predicts zero-shot robustness to early exit and layer skipping without retraining.
4. **Scaling Analysis and Best Practices:** We analyze performance across model and data scales, recommending a progressively increasing distribution across *depth* paired with a decreasing schedule across *steps*, yielding up to 25% training FLOPs savings and up to 1.5× inference speedup.

2 Related Work

Dropout granularity and scope. Dropout encompasses a family of techniques that differ in the granularity at which stochastic sparsity is applied. Prior work distinguishes activation-level dropout, weight-level dropout (e.g., DropConnect (Wan et al., 2013)), and structured dropout that operates on groups of parameters such as channels, layers, or blocks (Salehin & Kang, 2023). In this paper, we focus exclusively on *structured, depth-wise dropout*—i.e., stochastic removal of entire transformer blocks during training—commonly referred to as *layer dropout* or *stochastic depth* (Huang et al., 2016). We do not study neuron-level or weight-level

dropout, which induce fine-grained sparsity and are known to interact differently with hardware efficiency and optimization dynamics.

Dropout in large-scale LLM pretraining. As language models scaled to billions of parameters and trillion-token datasets, explicit regularization techniques—including activation dropout—have largely disappeared from state-of-the-art pretraining recipes. Early decoder-only models such as GPT-3 (Brown et al., 2020) and OPT (Zhang et al., 2022) retained the dropout settings inherited from Vaswani et al. (2017), while later models such as PaLM (Chowdhery et al., 2022) applied dropout only during finetuning, and LLaMA-style models no longer explicitly document its use. Recent empirical studies further suggest that activation dropout can degrade performance in single-epoch, large-data regimes (Liu et al., 2025), reinforcing the prevailing view that dropout is unnecessary or harmful at scale. At the same time, dropout has been shown to remain beneficial in multi-epoch or data-limited settings (Xue et al., 2023), indicating that its utility is highly regime-dependent.

Importantly, techniques originally introduced as regularizers may persist in modern LLM training for reasons unrelated to overfitting prevention. Weight decay, for example, has been shown to primarily influence optimization dynamics rather than classical generalization in large-scale pretraining (D'Angelo et al., 2024). This motivates re-examining dropout—particularly structured variants—without assuming that its value must stem from regularization in the traditional sense.

Layer dropout across scale. Layer dropout was originally proposed to stabilize optimization in very deep residual networks (Huang et al., 2016) and later become standard in large-scale vision models. However, its optimal strength has been observed to diminish as dataset scale increases: for example, ConvNeXt models trained on ImageNet-22K require substantially lower dropout rates than those trained on ImageNet-1K (Liu et al., 2022). A similar pattern appears in language modeling. Progressive Layer Dropout (Zhang & He, 2020) and LayerDrop (Fan et al., 2020) reported improved convergence and robustness in BERT-era, multi-epoch settings on relatively small corpora. In contrast, more recent work applying layer dropout to decoder-only LLMs trained on large token budgets has reported non-negligible accuracy degradation (Elhoushi et al., 2024), suggesting that naive extensions of earlier recipes may not transfer to modern regimes. To date, the literature lacks a controlled, large-scale evaluation that reconciles these conflicting findings by systematically varying dropout configurations and optimizer settings.

Training-aware approaches to depth elasticity. Layer dropout is closely related to a broader class of training-aware methods designed to enable inference-time efficiency. In compression, approaches such as Quantization-Aware Training (QAT) consistently outperform post-training quantization by exposing the model to reduced precision during optimization (Stock et al., 2021). Analogously, depth-aware training aims to make models robust to reduced depth at inference time. Prior work has explored achieving depth elasticity via auxiliary losses, routers, or adapters added during or after pretraining, including early-exit models (Jamialahmadi et al., 2025), routing-based skipping (Jiang et al., 2024; Raposo et al., 2024), and hybrid speculative decoding schemes (Zhang et al., 2024a). Other approaches train elastic architectures explicitly, such as Once-for-All (Cai et al., 2020), MatFormer (Devvrit et al., 2024), and Nemotron-Elastic (Taghibakhshi et al., 2025). While effective, these methods typically introduce architectural changes, additional parameters, or auxiliary objectives.

Layer dropout occupies a distinct position within this landscape: it induces robustness to depth-wise inference optimizations directly during pretraining, without modifying the model architecture or introducing additional losses. Prior work demonstrated that this can enable elastic inference at small scales (Fan et al., 2020), but whether similar benefits can be realized at modern LLM scales without sacrificing base-model accuracy has remained unresolved.

3 Methodology

In our experiments, we train decoder-only transformers following the architecture of Celerity models (Bergsma et al., 2025b): ALiBi position embeddings (Press et al., 2022), squared ReLU activations (Zhang et al., 2024b),

and Llama3 vocabulary (Grattafiori et al., 2024). Specific architectural dimensions for all model sizes are detailed in the Appendix. Our datasets are obtained from a diverse corpus of natural language text and code.

To develop best practices and quantify the effects of layer dropout, we first identify optimal hyperparameters for each dropout rate (Sec. 5). We then determine optimal granularity (Sec. 6) and configuration (Sec. 7). Following Hoffmann et al. (2022), these experiments utilize a compute-optimal budget of 20 tokens-per-parameter (TPP) at each model size. Subsequently, we evaluate benefits across various depth-wise inference optimizations (Sec. 8), then quantify accuracy as training scales to larger datasets (Sec. 9). We conclude with larger-scale runs with aggressive dropout rate to demonstrate its final performance and inference advantages.

4 Preliminary

General Formulation of Layer Dropout We start by denoting residual layer $\ell \in \{0, \ldots, L-1\}$, of an L layer neural network at training step $t \in \{0, \ldots, T-1\}$, as:

$$\mathbf{H}^{\ell+1,t} = \mathbf{H}^{\ell,t} + f^{\ell}(\mathbf{H}^{\ell,t}) \tag{1}$$

where, in the domain of natural language processing, activation tensor $\mathbf{H} \in \mathbb{R}^{B \times S \times d}$, B is batch size, S is sequence length, d is hidden dimension.

When layer dropout is applied with rate $p^{l,t}$, the operation of the layer during training at step t becomes:

$$\mathbf{H}^{\ell+1,t} = \mathbf{H}^{\ell,t} + r^{\ell,t}_{\text{train}} \mathbf{M}^{\ell,t} f^{l}(\mathbf{H}^{\ell,t}) \tag{2}$$

where mask $\mathbf{M}^{\ell,t} \in \{0,1\}^{B} \sim \text{Bernoulli}(1 - p^{\ell,t})$ is a Bernoulli random vector, and r_{train} is a scaling factor applied during training. r_{train} is defined differently in different layer dropout literature, and we will discuss our choice later.

The b^{th} sequence of $\mathbf{H}$ during training is now equal to[1]:

$$\mathbf{H}^{\ell+1,t}[b,:,:] = \begin{cases} \mathbf{H}^{\ell,t}[b,:,:], \\ \qquad \text{with probability } p, \\ \mathbf{H}^{\ell,t}[b,:,:] + r^{\ell,t}_{\text{train}} f^{l}\big(\mathbf{H}^{\ell,t}[b,:,:]\big), \\ \qquad \text{with probability } 1-p. \end{cases} \tag{3}$$

While layer dropout could be implemented during training by executing $f(\mathbf{H}^{\ell,t})$ on all sequences $b \in \{0, 1, ..., B-1\}$ of $\mathbf{H}$, and multiplying its output by $\mathbf{M}^{\ell,t}$, a more efficient implementation would be to only execute $f(\mathbf{H}^{\ell,t})$ on sequences $b \in \{\, b_i \mid \mathbf{M}^{\ell,t}[b_i] = 1 \,\}$. This leads to a saving a portion p of training FLOPs of the layer.

During inference, dropout is typically disabled and a distinct scaling factor, r^{ℓ}_{eval}, is applied:

$$\mathbf{H}^{\ell+1} = \mathbf{H}^{\ell} + r^{\ell}_{\text{eval}} f^{\ell}(\mathbf{H}^{\ell}) \tag{4}$$

Layer Dropout for a Transformer We denote the operation of layer $\ell \in \{0, \ldots, L-1\}$ of an L, layer transformer model, at time step $t \in \{0, \ldots, T-1\}$, during training as:

$$\begin{aligned} \mathbf{Z}^{\ell,t} &= \mathbf{X}^{\ell,t} + f^{l}_{\text{attn}}(\mathbf{X}^{\ell,t}) \\ \mathbf{X}^{\ell+1,t} &= \mathbf{Z}^{\ell,t} + f^{l}_{\text{ffn}}(\mathbf{Z}^{\ell,t}) \end{aligned} \tag{5}$$

where $\mathbf{X}, \mathbf{Z} \in \mathbb{R}^{B \times S \times d}$, f^{ℓ}_{attn} is the attention layer and f^{ℓ}_{ffn} is the feed-forward network (FFN).[2]

[1] For neuron dropout, i.e., the default variant of dropout introduced by (Hinton et al., 2012), $\mathbf{M} \in \{0,1\}^{\{B \times S \times d\}}$.

[2] This is a simplified form that does not refer to layer normalization or different variants of attention and FFN, but the subsequent formulation generalizes to different transformer variants that include pre-, post-, layer normalization, different variants or alternatives to attention, FFNs, and mixture of experts, as long as residual connection exists.

When layer dropout is applied with rate $p^{\ell,t}$, the operation at transformer layer, ℓ, step, t, during training becomes:

$$\begin{aligned}\mathbf{Z}^{\ell,t} &= \mathbf{X}^{\ell,t} + r^{\ell,t}_{\text{train}}\mathbf{M}^{\ell,t}_{\text{attn}} f^{\ell}_{\text{attn}}(\mathbf{X}^{\ell,t}) \\ \mathbf{X}^{\ell+1,t} &= \mathbf{Z}^{\ell,t} + r^{\ell,t}_{\text{train}}\mathbf{M}^{\ell,t}_{\text{ffn}} f^{\ell}_{\text{ffn}}(\mathbf{Z}^{\ell,t})\end{aligned} \tag{6}$$

and during inference becomes:

$$\begin{aligned}\mathbf{Z}^{\ell,t} &= \mathbf{X}^{\ell,t} + r^{\ell,t}_{\text{eval}} f^{\ell}_{\text{attn}}(\mathbf{X}^{\ell,t}) \\ \mathbf{X}^{\ell+1,t} &= \mathbf{Z}^{\ell,t} + r^{\ell,t}_{\text{eval}} f^{\ell}_{\text{ffn}}(\mathbf{Z}^{\ell,t})\end{aligned} \tag{7}$$

5 Hyperparameters

Background To avoid the "hyperparameter lottery" phenomenon (Dey et al., 2024), and to ensure we compare against a strong baseline, we systematically optimize learning rate, batch size, and weight decay for each dropout rate before evaluating configurations. Prior literature offers varying strategies—from coupling dropout with *max*-norm regularization (Hinton et al., 2012) to using learning rates 10× larger than baselines (Zhang & He, 2020)—yet systematic consensus remains elusive. To our knowledge, this is the first study to perform joint optimization of these hyperparameters for layer-wise dropout. We tune a small model with dimensions depth L_{base}, width d_{base} on dataset D_{base} to determine learning rate η_{base}, weight decay λ_{base}, initialization σ_{base}, ans batch size B_{base}, then scale via μP (Yang & Hu, 2021), CompleteP (Dey et al., 2025), and Power Lines (Bergsma et al., 2025a).

Dropout Scale The scaling parameters r_{train} and r_{eval} from Equations 2 and 4 require careful consideration. We define layer density $\rho = 1 - p$. The choice of scaling parameters varies across different research work and frameworks. Moreover, they are occasionally left implicit in published papers, and we often need to inspect their source code to specify which scaling they use. The original dropout paper (Hinton et al., 2012) used $r_{\text{train}} = 1$, $r_{\text{eval}} = \rho$. Standard libraries like PyTorch and TensorFlow use $r_{\text{train}} = 1/\rho$ for dropout. For layer dropout, the first stochastic depth paper (Huang et al., 2016) used $r_{\text{train}} = 1$, $r_{\text{eval}} = p$; DINOv2 (Oquab et al., 2024)[3] used $r_{\text{train}} = 1/\rho$, $r_{\text{eval}} = 1$; `fairseq`[4] (that implemented Fan et al. (2020)) and `torchtune`[5] (that implemented Elhoushi et al. (2024)) set both to 1. We demonstrate that selecting $r_{\text{train}} = 1/\rho$ is critical for stable hyperparameter transfer.

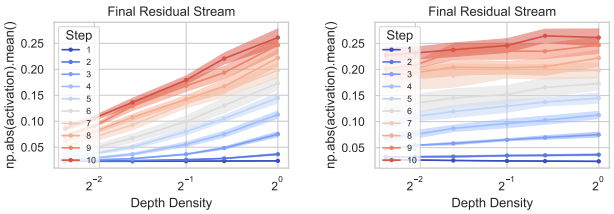
To determine the optimal scale factor r_{train}, we follow CompleteP's Maximal Residual Stream Update Desideratum (Dey et al., 2025), which facilitates hyperparameter transfer across model depths L.

Desideratum 1 (Maximal Residual Stream Update)**.** Each residual block's weights should contribute order $1/L$ to feature movements, and each non-residual block should contribute constant order. More precisely, for all $\ell \in [L-1]$, each block's parameter update $\boldsymbol{\theta}^{\ell} \mapsto \boldsymbol{\theta}^{\ell} + \Delta\boldsymbol{\theta}^{\ell}$ should contribute the change $\frac{1}{d}\|\Delta_{\boldsymbol{\theta}^{\ell}}\mathbf{H}^{\ell+1}\|_2^2 \in \Theta(1/L)$. Moreover, for the embedding and unembedding layers we require $\frac{1}{d}\|\Delta\mathbf{W}^0\mathbf{X}\|_2^2 \in \Theta(1)$ and $\frac{1}{d}\|\Delta\mathbf{W}^L\mathbf{H}^L\|_2^2 \in \Theta(1)$.

Since layer dropout reduces the effective depth of the network during training, we treat models with different dropout rates ρ as having different effective depths, and apply this desideratum to ensure stable initialization across these effective depths. Our coordinate checks in Fig. 2 empirically evaluate which scaling factor better satisfies stable initialization across dropout rates: $r_{\text{train}} = 1$ fails, necessitating per-rate tuning, whereas $r_{\text{train}} = 1/\rho$ largely satisfies these checks, enabling optimal hyperparameters transfer across many layer dropout rates.

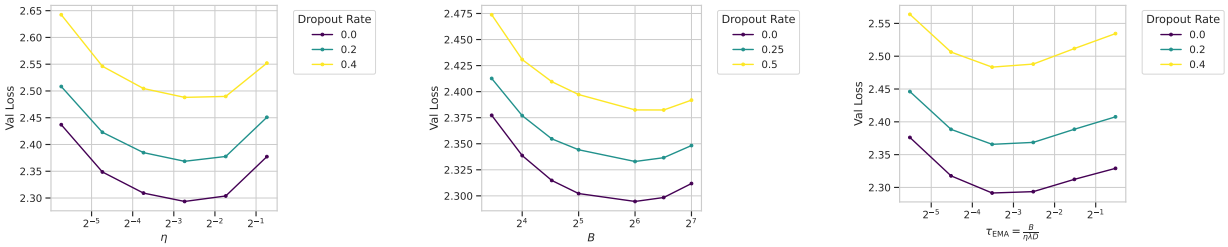
Transfer Test Fig. 3 verifies that $r_{\text{train}} = 1/\rho$ enables hyperparameter transfer: optimal η, λ, and B remain constant across dropout rates. Hence, we adopt Table A.2's transfer rules with $r_{\text{train}} = 1/\rho$ for all our upcoming experiments. We set $r_{\text{eval}} = 1$ to ensure that $\mathbf{H}^{\ell+1}_{\text{eval}} = \mathbb{E}[\mathbf{H}^{\ell+1}_{\text{train}}]$.

[3] https://github.com/facebookresearch/dinov2/blob/main/dinov2/layers/drop_path.py
[4] https://github.com/facebookresearch/fairseq/blob/main/fairseq/modules/layer_drop.py
[5] https://github.com/meta-pytorch/torchtune/blob/main/torchtune/modules/layer_dropout.py



(a) $r^{l,t}_{\text{train}} = 1$ (b) $r^{l,t}_{\text{train}} = 1/\rho^{l,t}$ 👑

Figure 2: Coordinate Check. Scaling with $r^{l,t}_{\text{train}} = 1/\rho^{l,t}$ during training with layer dropout yields stable activation scale across depth density. More details in Sec. B.



(a) Learning rate. (b) Batch size. (c) AdamW time scale.

Figure 3: Analysis of hyperparameter transferability on 271M model. We observe that optimal value for each hyperparameter remains similar across most layer dropout rates with the scaling factor $r_{\text{train}} = 1/\rho$.

6 Dropout Granularity

6.1 Model Granularity

Background When layer dropout was first introduced by (Huang et al., 2016), it was applied on residual blocks in CNNs, where each residual block consisted of two convolution-batchnorm pairs, separated by ReLU. In transformers, each layer consists of 2 residual blocks: an attention residual block followed by a FFN residual block. An open question is whether to apply layer dropout separately to attention and FFN (i.e., the Bernoulli mask tensors $\mathbf{M}^{\ell,t}_{\text{attn}}$ and $\mathbf{M}^{\ell,t}_{\text{ffn}}$ are sampled independently at each training step, t), which we refer to as Sub-Layer Dropout, or to apply it on the whole transformer layer (i.e., $\mathbf{M}^{\ell,t}_{\text{attn}} = \mathbf{M}^{\ell,t}_{\text{ffn}} \quad \forall \ell$), which we refer to as Layer Dropout. Different research work have used different types: DINOv2 (Oquab et al., 2024) and (Zhang & He, 2020) used sub-layer dropout, while LayerDrop (Fan et al., 2020) and LayerSkip (Elhoushi et al., 2024) used layer dropout. However, to the best of our knowledge, we are the first to systematically evaluate a comparison between them.

Analysis In Table 1 we compare layer dropout and sub-layer dropout at various model sizes. The results clearly show that in terms of accuracy, Layer Dropout is better. Note that as model size increases, loss degradation introduced by dropout diminishes, which will later encourage us to try larger dropout rates for larger models. This may be contrary to the notion that finer grain sparsity leads to higher accuracy, but could be explained by other research work that show that attention and FFN work in tandem (Agarwal et al., 2026). We leave investigating the reason sub-layer dropout underperforms layer dropout, and also leave investigating other configurations such as applying dropout only on attention or only on FFN, for future work.

Finding 1: Layer dropout that drops whole transformer blocks for each sample, leads to higher accuracy results than sub-layer dropout that drops attention and FFN sub-blocks separately.

Table 1: Ablating model granularities. Models trained at 20 TPP.

Model Size	Dropout Type	Dropout Rate	Train Loss ↓	% Δ ↓	Val Loss ↓	% Δ ↓
271M	Baseline	—	2.293	0.00%	2.294	0.00%
	SubLayer	0.1	2.421	4.72%	2.377	3.61%
	Layer 👑	0.1	**2.419**	**3.95%**	**2.367**	**3.18%**
503M	Baseline	—	2.140	0.00%	2.110	0.00%
	SubLayer	0.1	2.260	3.94%	2.174	3.03%
	Layer 👑	0.1	**2.178**	**2.60%**	**2.170**	**2.84%**

6.2 Tensor Granularity

Background The next open question we tackle is whether it is better to apply layer dropout at batch granularity, i.e. $\mathbf{M}^{\ell,t}[b] = M^{\ell,t}\ \forall b$, where $M^{\ell,t} \sim \text{Bernoulli}(1-p^{\ell,t})$ is drawn once per layer ℓ and step t (so $\mathbf{M}^{\ell,t}[b]$ takes the same value for all sequences b), or at sequence granularity, i.e. $\mathbf{M}^{\ell,t}[b] \sim \text{Bernoulli}(1-p^{\ell,t})$ drawn i.i.d. for each b (so $\mathbf{M}^{\ell,t}[b]$ is sampled independently for each sequence b).

In literature, this does not seem to have been discussed, and we usually need to resort to the codebases of different papers to find out which type each has used. The implementation of the pioneer Stochastic Depth paper[6] as well as the `fairseq`[7] implementation of LayerDrop used per-batch layer dropout, while DINOv2 (Oquab et al., 2024)[8], `timm`[9], and `torchtune`[10] implementation of LayerSkip used per-sequence. To the best of our knowledge, we are the first to systematically compare per-batch and per-sequence layer dropout.

Analysis Fig. 4 compares the accuracy results of applying layer dropout per batch and per sequence. The results clearly show that per-sequence leads to better losses. This is in line with the notion that finer grain sparsity leads to higher accuracy. In terms of compute performance, per-batch layer dropout has the advantage of not having to load the weights of a layer during a training step. However, if training is compute bound (i.e., batch size and context length are large enough), per-sequence dropout should lead to speedup similar to per-batch dropout as both save the same compute FLOPs. A middle ground that could combine the benefits of not loading weights of per-batch dropout and fine-grain sparsity of per-sequence dropout, could be satisfied in distributed training where each device drops different batches, or training with gradient accumulation where a different mini-batch is dropped per gradient accumulation step. We leave exploring such approaches, as well as comparing with even finer-grain dropout such as per-token or per-neuron, for future work.

Finding 2: For any given layer dropout rate, dropout per-sequence leads to lower loss than dropout per-batch.

7 Dropout Configurations

7.1 Dropout Distribution

Background Various dropout distributions across layers have been proposed to optimize training efficiency and model depth. We formalize three primary distributions for dropout rate p at layer ℓ:

1. **Uniform Distribution**: where all layers have the same dropout rate, $p^{\ell,t}_{\text{uniform}} = p_{\max}$.

[6] https://github.com/yueatsprograms/Stochastic_Depth/blob/master/ResidualDrop.lua
[7] https://github.com/facebookresearch/fairseq/blob/main/fairseq/modules/layer_drop.py
[8] https://github.com/facebookresearch/dinov2/blob/main/dinov2/layers/drop_path.py
[9] https://github.com/huggingface/pytorch-image-models/blob/main/timm/layers/drop.py
[10] https://github.com/meta-pytorch/torchtune/blob/main/torchtune/modules/layer_dropout.py

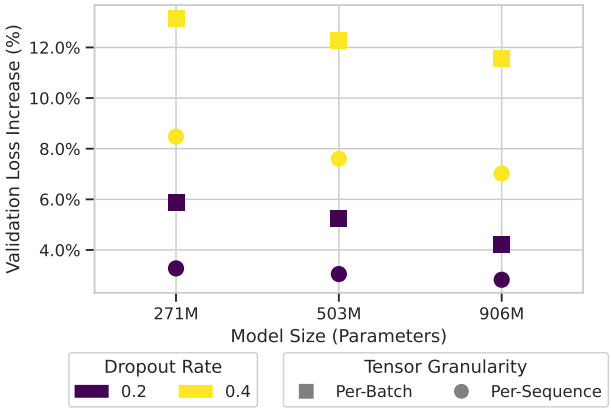


Figure 4: Ablating tensor granularity. Models trained at 20 TPP.

2. **Increasing Layer Distribution (ILD)**: where dropout rate starts at 0 at the first layer and linearly increases across layers to reach $p_{\max}$ at the last layer, $p^{\ell,t}_{\text{ILD}} = \frac{\ell}{L-1} \cdot p_{\max}$ (Huang et al., 2016; Oquab et al., 2024; Zhang & He, 2020; Elhoushi et al., 2024).
3. **Alternating Layer Distribution (ALD)**: where layer dropout is only applied at every other layer, $p^{\ell,t}_{\text{ALD}} = p_{\max} \cdot \mathbf{1}_{\ell \equiv 1 (\text{mod } 2)}$ (Fan et al., 2020).

where $p_{\max}$ is the maximum dropout rate. Note that $p^{\ell,t}_{\text{ALD}} \in \{0, p_{\max}\}$, whereas $p^{\ell,t}_{\text{ILD}} \in [0, p_{\max}]$.

For any distribution, the average dropout and corresponding nonembedding FLOPs[11] savings at step t are defined as:

$$p^t_{\text{mean}} = \text{FLOPs Savings}^t = \frac{1}{L} \sum_{\ell=0}^{L-1} p^{\ell,t} \tag{8}$$

Mathematically, $p^t_{\text{ILD}_{\text{mean}}} = 0.5 p_{\max}$[12] and $p^t_{\text{ALD}_{\text{mean}}} = \frac{\lfloor L/2 \rfloor}{L} p_{\max}$, which is $\approx 0.5 p_{\max}$ for typical L. To the best of our knowledge, this study is the first to systematically analyze the differences between these layer dropout distributions at fixed FLOPs budgets.

Analysis In Table 2, we compare uniform, ILD, and ALD grouped by equivalent FLOPs savings, finding that non-uniform distributions consistently outperform uniform ones under a fixed average dropout (as well as fixed FLOPs budget). While ALD is superior at the smallest model size, its advantage diminishes with scale, whereas ILD's improvement over uniform widens. Although our ALD results with $p_{\max} = 0.2$ do not beat the baseline as reported in the multi-epoch regime of LayerDrop (Fan et al., 2020), the observed reduction in dropout-induced degradation as models grow encourages further investigation at larger scales.

Finding 3: For the same training FLOPs budget, non-uniform dropout distribution across layers is better than uniform. As a model scales, ILD is recommended.

7.2 Dropout Schedule

Background While distributions govern sparsity across depth, the temporal schedule determines how regularization pressure evolves throughout pre-training. Hillier et al. (2024) found decreasing schedules were better for LLM pre-training, whereas increasing schedules were superior for fine-tuning; however, Liu et al. (2025) recently claimed both fail in modern regimes. We formalize various time schedules for dropout rate p at step t over total duration T, where p^{ℓ}_{dist} represents a chosen layer distribution:

1. **Constant Time Schedule**: where dropout rate is constant throughout training steps, $p^{\ell,t}_{\text{constant}} = p^{\ell}_{\text{dist}}$.

[11] For the remaining of the paper, we use the term FLOPs to refer to nonembedding FLOPs.

[12] Follows from applying the arithmetic series formula $\sum_{i=0}^{n-1} a_i = \frac{n}{2}(a_0 + a_{n-1})$ to the per-step mean $p^t_{\text{ILD}_{\text{mean}}} = \frac{1}{L}\sum_{\ell=0}^{L-1} p^{\ell,t}_{\text{ILD}} = \frac{1}{L}\sum_{\ell=0}^{L-1} \frac{\ell}{L-1} \cdot p_{\max} = \frac{p_{\max}}{L} \cdot \frac{L}{2}\left(\frac{0}{L-1} + \frac{L-1}{L-1}\right) = \frac{1}{2} p_{\max}$.

Table 2: Analysis of Dropout Distributions across layers. Models trained at 20 TPP.

Model	Training FLOPs Savings	Max Rate	Dropout Distb.	Val	% Δ
271M	0%	-	-	2.294	0.00%
	10%	0.1	Uniform	2.331	1.62%
		0.2	Alternating 👑	**2.323**	**1.29%**
		0.2	Increasing	2.328	1.52%
	20%	0.2	Uniform	2.369	3.27%
		0.4	Alternating 👑	**2.357**	**2.76%**
		0.4	Increasing	2.363	3.01%
503M	0%	-	-	2.110	0.00%
	10%	0.1	Uniform	2.141	1.48%
		0.2	Alternating 👑	2.143	1.57%
		0.2	Increasing	**2.132**	**1.08%**
	20%	0.2	Uniform	2.174	3.05%
		0.4	Alternating	2.169	2.82%
		0.4	Increasing 👑	**2.162**	**2.48%**
906M	0%	-	-	1.953	0.00%
	10%	0.1	Uniform	1.977	1.26%
		0.2	Alternating	1.979	1.36%
		0.2	Increasing 👑	**1.972**	**1.01%**
	20%	0.2	Uniform	2.008	2.82%
		0.4	Alternating	2.004	2.66%
		0.4	Increasing 👑	**1.998**	**2.34%**

2. **Increasing Time Schedule (ITS)**: where dropout rate starts at 0 at the beginning of training and linearly increases to p^{ℓ}_{dist} at the end of training, $p^{\ell,t}_{\text{ITS}} = p^{\ell}_{\text{dist}} \cdot \left(\frac{t}{T-1}\right)$.
3. **Decreasing Time Schedule (DTS)**: where dropout rate starts at p^{ℓ}_{dist} at the beginning of training and linearly decreases to 0 at the end of training, $p^{\ell,t}_{\text{DTS}} = p^{\ell}_{\text{dist}} \cdot \left(1 - \frac{t}{T-1}\right)$.

To compare these schedules fairly, we define the mean training dropout $\bar{P}$ as the average rate across depth and time, representing total active training FLOPs savings:

$$\bar{P} = \text{FLOPs Savings}_{\text{total}} = \frac{1}{T}\sum_{t=0}^{T-1}\left(\frac{1}{L}\sum_{\ell=0}^{L-1} p^{\ell,t}\right) \tag{9}$$

Consequentially, $\bar{P}_{\text{uniform,ITS}} = 0.5p_{\max}$ and $\bar{P}_{\text{ILD,ITS}} = \bar{P}_{\text{ILD,DTS}} = 0.25p_{\max}$[13].

Analysis In Table 3, we group configurations by total FLOPs savings. Across all scales, decreasing schedules consistently outperforms constant and increasing schedules. Notably, at 5% FLOPs savings for 503M & 906M, combined ILD and DTS achieves lower validation losses than the dense baseline, demonstrating for the first time that it is possible to beat the dense baseline with fewer training FLOPs.

[13] $\bar{P}_{\text{ILD,DTS}} = \frac{1}{LT}\sum_{t=0}^{T-1}\sum_{l=0}^{L-1} p^{l,t}_{\text{ILD,DTS}}$, where $p^{l,t}_{\text{ILD}} = p^{t}\frac{l}{L-1}$ and $p^{l,t}_{\text{DTS}} = p^{l}\left(1 - \frac{t}{T-1}\right)$. Separating the double sum into independent factors and applying the arithmetic series formula $\sum_{i=0}^{n-1} a_i = \frac{n}{2}(a_0 + a_{n-1})$ to each, the inner sum over l evaluates to $\frac{L}{2}(0+1)\cdot p_{\max}$ and the outer sum over t evaluates to $\frac{T}{2}(1+0)$, yielding the closed form $P_{\text{ILD,DTS}} = \frac{1}{4}\cdot p_{\max}$.

Training FLOPs Savings	Max Dropout	Layer Dist.	Time Sched.	271M		503M		906M	
				Val	% Δ	Val	% Δ	Val	% Δ
0%	–	–	–	2.294	0.00%	2.110	0.00%	1.953	0.00%
5%	0.05	–	–	2.311	0.77%	2.124	0.67%	1.963	0.54%
	0.1	Increasing	–	2.310	0.73%	2.121	0.56%	1.961	0.42%
	0.2	Increasing	Increasing	2.325	1.36%	2.139	1.38%	1.979	1.34%
	0.2	Increasing	Decreasing 👑	**2.304**	**0.46%**	**2.110**	**0.03%**	**1.951**	**−0.06%**
10%	0.1	–	–	2.331	1.62%	2.141	1.48%	1.977	1.26%
	0.2	Increasing	–	2.328	1.52%	2.132	1.08%	1.972	1.01%
	0.4	Increasing	Increasing	2.357	2.75%	2.172	2.95%	2.014	3.15%
	0.4	Increasing	Decreasing 👑	**2.312**	**0.82%**	**2.116**	**0.28%**	**1.955**	**0.10%**
20%	0.2	–	–	2.369	3.27%	2.174	3.05%	2.008	2.82%
	0.4	Increasing	–	2.363	3.01%	2.162	2.48%	1.998	2.34%
	0.8	Increasing	Increasing	2.442	6.46%	2.257	6.99%	2.090	7.04%
	0.8	Increasing	Decreasing 👑	**2.361**	**2.93%**	**2.147**	**1.75%**	**1.983**	**1.55%**

Table 3: Ablating dropout time schedule. We group different dropout configurations that have the same active non-embedding FLOPs reduction induced by layer dropout across all training steps. Models trained at 20 TPP.

Conversely, increasing schedule significantly degrades loss. While (Zhang & He, 2020) reported positive results with an exponential increasing schedule, we do not observe benefits in our large-scale single-epoch regime. We hypothesize the decreasing schedule's effectiveness stems from high initial noise at the beginning of training forcing weight space exploration (reducing bias), while subsequent decay allows settling into a stable minimum (reducing variance). This can also be viewed as a form of stochastic model growing, where effective capacity increases smoothly throughout training without explicit re-initialization of conventional model growing (e.g., (Samragh et al., 2024)). It can also be viewed as a form of curriculum learning (Wang et al., 2021), that starts training with a hard task of learning using small effective depth and the learning task gradually becomes easier as effective depth increases.

We leave other schedules such as applying dropout to mid-training, SFT, or continual pre-training for future work.

Finding 4: For a fixed training FLOPs budget, a schedule decaying from a maximum rate to zero consistently achieves the highest accuracy, outperforming both constant and increasing schedules across all scales.

Key takeaway 1: The best recommended practice for layer dropout configuration is increasing dropout across layers and decreasing dropout across time. This leads to the best accuracy for a given training FLOPs budget.

8 Inference Optimizations

A primary motivation for pre-training with layer dropout is to induce robustness to depth-wise optimizations, including early exit, layer skipping, and layer pruning. We explore techniques that keep pre-trained weights intact. We categorize such techniques into "Zero-Shot" inference approaches that merely apply autoregressive decoding inference on a model with fewer layers without any modifications, and "Post-Training" approaches that add adapters or routers (albeit not modifying the model's weights) or modify the inference decoding algorithm. We leave pruning approaches that require fine-tuning or weight modification, e.g., Xia et al. (2024); Lu et al. (2024), for future work.

8.1 Zero-Shot Inference Benefits

8.1.1 Early Exit

We define early exit at layer ℓ' as executing the embedding layer, transformer layers 0 to $\ell' - 1$, and the unembedding layer. This "static early exit" is equivalent to skipping layers ℓ' to $L - 1$. Fig. 5 and A.2 illustrate results for various models. We see that for a model trained without dropout, loss deteriorates significantly even after exiting one layer earlier, but for a model trained with dropout, loss remains steady when exiting early for a portion of layers. Throughout all layers, exiting at any layer for a model pretrained with dropout has lower loss than a model pretrained without, with lower loss for a model pretrained with higher dropout. Fig. A.2b shows that uniform and ILD exhibit early exit improvements, but ALD does not.

Does a decreasing schedule sacrifice depth robustness by ending training dropout-free? Fig. A.2b says no: models trained with decreasing schedules significantly outperform zero-dropout baselines at early exit, and Fig. 5 shows they match - and in some cases exceed - constant schedules when controlling for training FLOPs — proving that exposure to dropout during early training leaves lasting benefits.

Key takeaway 2: Pre-training with layer dropout not only saves training FLOPs but always leads to better early exit loss during inference.

Finding 5: For the same pre-training FLOPs budget, decreasing dropout schedule leads to the lowest validation loss as well as competitive early exit loss.

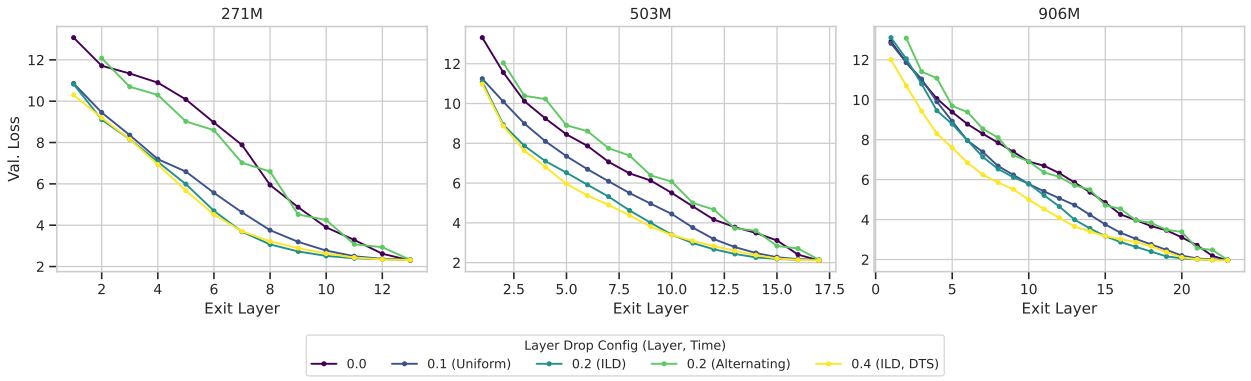


Figure 5: Early-exit validation loss for different model sizes trained at 20 TPP: no dropout vs. dropout configurations with 10% FLOPs savings. Additional comparisons in Fig. A.2.

Here, we have covered static early exit where all tokens exit at the same layer. We hypothesize that "dynamic early exit" where each token exits at a different layer based on a heurestic, router, or an auxiliary model (e.g., Schuster et al. (2022)), will lead to better accuracy-throughput tradeoffs on a model pretrained with layer dropout. However, we leave verifying this hypothesis for future work.

8.1.2 Intermediate Layer Skipping

Layer dropout induces structural robustness enabling models to function when layers are skipped at inference. As shown in Fig. A.3a, dense baselines exhibit immediate loss spikes when layers are skipped, whereas ALD facilitates graceful degradation. This zero-shot "elastic" effect allows a 906M model to bridge the gap toward smaller dense baselines, as shown in Fig. A.3b, providing flexibility typically requiring complex modifications and/or continual pretraining, like LlamaFlex (Cai et al., 2025) or Flextron (Cai et al., 2024), "for free" within the standard pre-training recipe.

Ablations at $p_{\max} = 0.2$ show that while all dropout variants improve skip-robustness, ALD offers superior retention for non-contiguous skipping (Fig. A.3c). Under iso-FLOP conditions (Fig. 6), ALD maintains lower

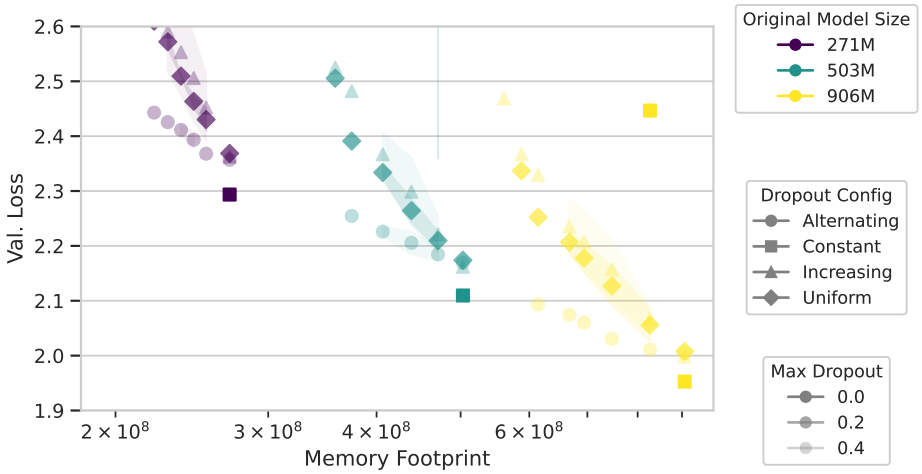


Figure 6: Intermediate layer skipping loss for models at 20 TPP: baseline vs. dropout configurations with 20% FLOPs savings. Extended results in Fig. A.3.

validation loss than ILD for an equivalent 20% training compute reduction, confirming ALD as optimal for depth-wise inference elasticity at a fixed budget.

A clear trade-off emerges: ALD excels at skip-robustness but fails at early-exit, while ILD achieves better base accuracy and early-exit robustness with sub-optimal skip capability. Practitioners should choose based on deployment needs, and consider ALD with extended training to recover baseline accuracy.

Key takeaway 3: Layer dropout induces an inherent elasticity that allows a larger model to gracefully degrade to the performance levels of smaller models when layers are skipped, providing a unified architecture for varying compute constraints.

Finding 6: Average training dropout rate predicts zero-shot robustness to early exit and layer skipping without retraining.

8.2 Post-Training Inference Benefits

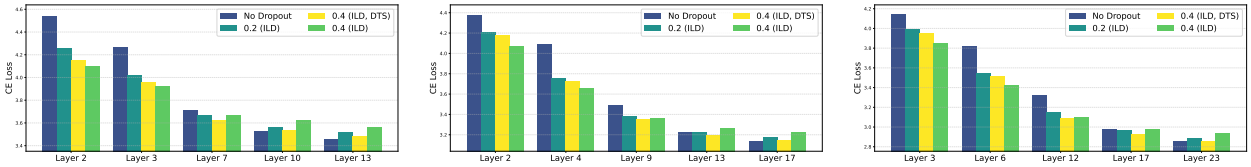
While zero-shot techniques exploit the inherent redundancy of a model, further efficiency gains can be achieved through targeted post-training modifications that do not alter the pre-trained weights. We define post-training benefits as those derived from secondary training phases—such as continual pre-training or fine-tuning—specifically focused on optimizing inference throughput. In this work, we limit our investigation to "weight-frozen" methods where the transformer backbone remains static, and optimization is achieved by training auxiliary modules like adapters or routers. This paradigm ensures that the model's foundational knowledge is preserved while expanding the Pareto-optimal frontier of its depth-wise flexibility. Here we cover using adapters and self-speculative decoding, and leave using routers (such as in Jiang et al. (2024)) for future work.

8.2.1 Early Exit Adapters

Background To evaluate if the structural benefits of layer dropout persist after supervised optimization, we utilize the *Balcony* framework for depth-based dynamic inference (Jamialahmadi et al., 2025). Balcony is a lightweight approach that freezes the pre-trained backbone and inserts additional transformer layers as "exit adapters" at selected points. These adapters are trained using a self-distillation objective where a Kullback–Leibler (KL) divergence loss aligns intermediate sub-model outputs with the final layer's predictions. While Jamialahmadi et al. (2025) demonstrates that incorporating these adapters directly into the pre-training

phase leads to even lower early-exit loss compared to post-training addition, such joint training increases the memory footprint and FLOPs per step, potentially slowing down the pre-training process.

Analysis As shown in Fig. 7, models pre-trained with layer dropout consistently outperform dense baselines across all model scales—270M, 503M, and 906M—even when exit adapters are only added post-training. Our proposed method offers a more efficient alternative to joint adapter pre-training: by incorporating layer dropout, we improve the loss of earlier layers without increasing the memory footprint or computational overhead during pretraining. In fact, layer dropout actively reduces training FLOPs while inducing a permanent structural robustness that auxiliary training can leverage but cannot fully replicate on a standard dense model. Furthermore, models trained with higher dropout rates demonstrate a superior "head-start" for adapter training, reaching lower validation losses at earlier layers than their low-dropout counterparts, confirming that depth-aware pre-training is a prerequisite for maximizing the efficacy of post-training strategies.



(a) 270M ($L = 13$) 20 TPP. (b) 503M ($L = 17$) 20 TPP. (c) 906M ($L = 23$) 20 TPP.

Figure 7: Early exit losses for models pre-trained with different dropout configurations, followed by freezing their weights and training early exit adapters as proposed by Balcony (Jamialahmadi et al., 2025). Models pre-trained with dropout always lead to better early exit losses even after adding exit adapters.

8.2.2 Self-Speculative Decoding

Background Speculative decoding accelerates autoregressive inference by using a fast "draft" model to predict tokens that are validated in parallel by a larger "target" model, enabling lossless speedup (Leviathan et al., 2023). Self-speculative methods, such as *Draft & Verify*, use a subset of the target model's own layers to act as the drafter (Zhang et al., 2024a). The effectiveness of this approach depends on identifying a subset of layers that is small enough for high throughput yet accurate enough to maintain high token acceptance rates.

Analysis We hypothesized that the structural elasticity induced by layer dropout enables the discovery of more efficient subsets. While Zhang et al. (2024a) used Bayesian optimization to find draft layers, we also apply other search methods: genetic algorithms, hill climbing, and simulated annealing, and select the search result that leads to highest speedup. Results in Table 4 confirm this showing models pre-trained with higher layer dropout obtain higher speedups during self-speculative decoding. For such models, search methods are able to find a subset of layers that achieve better trade offs of acceptance rate, α, and draft decoding time, T_{Draft}. We leave for future work the evaluation of other self-speculative techniques like LayerSkip (Elhoushi et al., 2024), which uses early exit for drafting, and Kangaroo (Liu et al., 2024), which employs early exit with adapters trained in a similar manner to Balcony (Jamialahmadi et al., 2025).

Model Size	TPP	Dropout Config.	Self-Speculative Decoding		
			Speedup	α	$\frac{T_{\text{Draft}}}{T_{\text{Target}}}$
270M	20	0	1.01×	92%	0.65
		0.2 (ILD)	1.20×	86%	0.55
		0.4 (ILD)	**1.35**×	89%	0.48
		0.4 (ILD, DTS)	1.22×	73%	0.57
504M	20	0	1.03×	90%	0.66
		0.2 (ILD)	1.22×	89%	0.53
		0.4 (ILD)	**1.43**×	97%	0.42
		0.4 (ILD, DTS)	1.20×	90%	0.52
906M	20	0	1.06×	94%	0.66
		0.2 (ILD)	1.32×	98%	0.49
		0.4 (ILD)	**1.40**×	97%	0.48
		0.4 (ILD, DTS)	1.29×	95%	0.52

Table 4: Self-speculative decoding speedup across model scales using Draft & Verify (Zhang et al., 2024a) on XSUM (Narayan et al., 2018). $\alpha \in [0, 1]$ is acceptance rate for draft length $\gamma = 5$, T_{Draft} is time to decode a single token for the selected subset of layers, and T_{Target} for the full model.

9 Scaling Analysis

In the preceding sections, we demonstrated that for compute-optimal pre-training at 20 TPP, specific layer dropout configurations—notably ILD+DTS—not only minimize degradation but can actually surpass the dense baseline in terms of validation accuracy. However, many ideas in architecture or pretraining in literature may perform well on small scale but fail to generalize at large scale, rendering them impractical to train foundational models. Hence, a critical question for modern foundational models is whether such benefits persist as the model is trained far beyond the compute-optimal point.

Inspired by the predictable scaling frameworks established for model size and data budget (Hestness et al., 2017; Kaplan et al., 2020; Hoffmann et al., 2022), and the recent investigation into how precision interacts with data scale (Kumar et al., 2024), we extend our analysis to high-TPP regimes. Our goal is to develop a scaling analysis that quantifies how the regularization and structural benefits of layer dropout evolve as the model exhausts its inherent redundancy through prolonged training. This allows us to predict the performance of sparse-trained models at the trillion-token scale typical of state-of-the-art LLMs.

Our recommended ILD+DTS configuration demonstrates that loss degradation remains remarkably stable as shown in Fig. 8, typically staying within $\approx 0.50\%$ of the baseline even at high TPP. These results indicate that layer dropout does not impede scaling performance. Instead, the stability observed suggests a robust, compute-efficient pre-training pathway that maintains structural benefits as we scale to training budgets of typical modern foundational LLMs.

10 Large-Scale Runs

To conclude the empirical evaluation, this section presents scaling of our optimized layer dropout recipe to models exceeding the 1B parameter threshold with aggressive dropout rates. This analysis serves to validate our primary hypotheses: that larger model architectures exhibit inherently higher robustness to structural sparsity and that aggressive dropout rates are the primary enabler for depth-wise inference flexibility.

Pushing the Limits of Structural Sparsity While previous studies often limited dropout rates to conservative values, such as 0.1 or 0.2 (Vaswani et al., 2017; Radford et al., 2019), we subject our 1.8B, 3.9B,

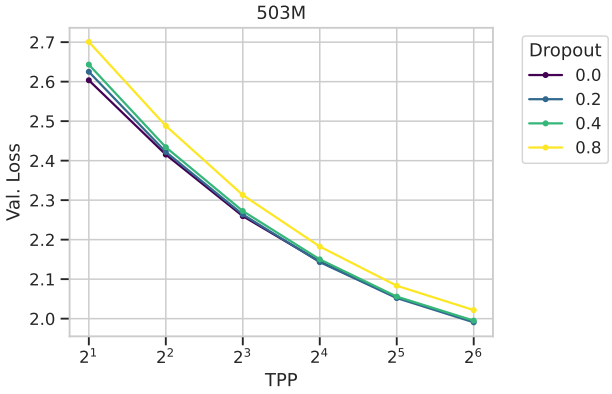


Figure 8: Validation loss across TPP for ILD with DTS, showing competitive performance with dense baselines as tokens-per-parameter increase.

Table 5: Training larger models with aggressive dropout preserves accuracy while accelerating training and inference.

Model Size	**1.8B**		**3.9B**		**8.2B**
TPP	20	20	20	20	20
Max Dropout	0	0.6	0	0.8	0.99
Layer Dist.	–	Inc.	–	Inc.	Inc.
Time Sched.	–	Dec.	–	Dec.	Dec.
FLOPs Savings ↑	0%	**15%**	0%	**20%**	**25%**
Val. Loss ↓	1.849	**1.836**	**1.732**	1.745	1.663
Skip Alt. Layers ↓	4.260	**2.282**	6.446	**2.129**	**1.991**
Early Exit @ $0.75L$ ↓	3.943	**2.329**	3.834	**2.143**	**1.777**
Spec. Decode Speedup ↑	1.10×	**1.34**×	1.02×	**1.54**×	**1.55**×

8.2B models to aggressive regimes with $p_{\max}$ values of 0.6, 0.8, and 0.99, respectively. Taking 3.9B model as an example, the training initialization is significant: the effective depth of the model begins at only $0.6L$[14], with the final layer being skipped 80% of the time. Despite this substantial reduction in early-training active

[14]Effective depth of a model with L layers at iteration t for ILD+DTS configuration is $\sum_{\ell=0}^{L-1}(1-p_{max}\frac{\ell}{L-1}) = \frac{1+1-p_{max}}{2}L = (1-0.5p_{max})L$.

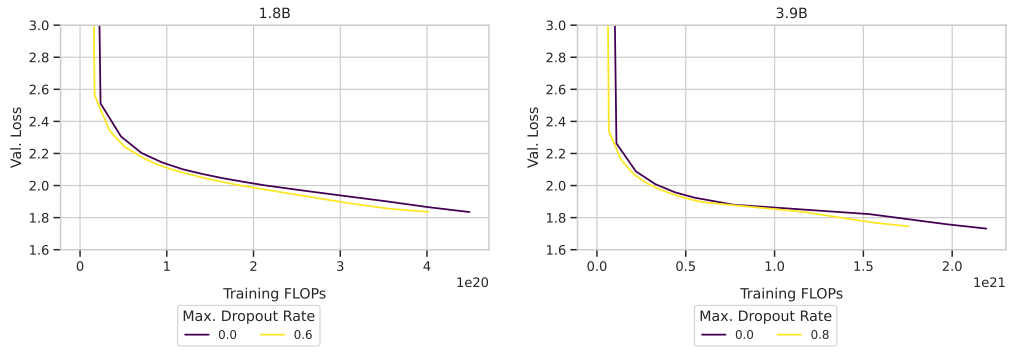


Figure 9: Validation loss versus training FLOPs for larger model sizes at aggressive maximum dropout rates for 20 TPP. At the same training FLOPs, training with layer dropout leads to better loss.

capacity, the models converge to results that are competitive with, or superior to, the dense baselines in both validation loss (as shown in Table 5) and downstream task performance (as shown in Table A.3).

Furthermore, Figure 9 shows that training with dropout is not only faster, but also achieves lower loss for a given FLOP budget throughout most of training under aggressive maximum dropout rates. Based on trends across model sizes, we project that larger models will exhibit even greater robustness to high dropout rates, yielding increasingly pronounced validation loss improvements over dense baselines as scale increases.

Unlocking Inference Efficiency The operational advantages of this depth-aware pre-training are most evident in the inference benchmarks in Table 5. The 3.9B model exhibits a 1.54× speedup in self-speculative decoding, a task where the dense baseline fails to provide significant gain, resulting in a 1.02× regression. This confirms that the elasticity induced by aggressive dropout is a prerequisite for effective speculative drafting at these scales. Furthermore, the model shows high resilience to static pruning; skipping alternate layers results in a cross-entropy loss of 2.129 for the dropout-trained model, whereas the dense baseline's loss increases to 6.446. Figures A.2c and A.3d show detailed results for skipping intermediate layers and early exit of those models.

Key takeaway 4: At the multi-billion parameter scale, aggressive layer dropout ($p_{\text{max}} = 0.99$) facilitates up to 25% savings in cumulative training FLOPs while largely preserving baseline generalization performance, unlocking zero-shot inference speedups of up to 1.55×.

11 Limitations

While our results establish a robust framework for layer dropout at scale, our work has multiple limitations:

- **Hyperparameter Transfer:** In our plots in Fig. 3, although optimal learning rate, η, and weight decay, λ, remained largely the same for small to medium dropouts, they reduced for aggressively high dropouts. Improving transferrability to such high dropouts would improve our accuracy results further. Our transfer analysis was primarily validated for constant dropout schedules. Extending these rules to the decreasing schedules identified as optimal could potentially yield further accuracy improvements at high TPP budgets.

- **Alternative Granularities:** We focused exclusively on transformer-level dropout to induce depth-wise robustness. Whether other granularities, such as attention-head or neuron-level dropout, can induce similar structural resilience remains an open question.

- **Comparison with Learned Depth Optimization:** This study did not compare structured layer dropout against learned depth-aware mechanisms such as Mixture-of-Depths (Raposo et al., 2024). Investigating the trade-offs between stochastic layer removal and dynamic, routing-based depth optimization is a compelling direction for future work.

- **Cross-Architecture Generalization:** While we validated results up to 8.2B parameters, the interaction between aggressive layer dropout and alternative architectures, such as Mixture-of-Experts (MoE) or non-transformer models, has not yet been explored.

- **Scaling Laws for Maximum Dropout:** We have not yet developed comprehensive scaling laws to predict the maximum dropout rate (p_{max}) that can be applied without incurring accuracy degradation relative to the dense baseline.

- **Scaling Analysis for Inference Benefits:** We have not quantified the different inference benefits (early exit loss, skipping intermediate layer loss, early exit adapter loss, and self-speculative decoding speedup) as TPP increases.

12 Conclusion

In this study, we have demonstrated that layer dropout is not only a viable technique for the modern large-scale pre-training regime but a powerful mechanism for architectural flexibility. By systematically exploring dropout distributions and schedules, we have shown that a depth-aware training approach—specifically utilizing an increasing distribution across layers coupled with a decreasing schedule over time—can maintain, and in some cases surpass, dense baseline accuracy while significantly reducing training FLOPs, and unlocking zero-shot inference elasticity and speedups. This approach is non-invasive and orthogonal to existing architectural and training optimizations, making it easily adoptable in standard LLM pre-training stacks.

Our "Hero Runs" on 1.8B, 3.9B, and 8.2B models reveal that as architectures grow larger, their natural resilience to aggressive dropout rates grow, with our recommended recipe unlocking zero-shot inference speedups of up to 1.55×. This inherent elasticity bridges the performance gap between discrete model sizes and enables high-efficiency deployment strategies, such as self-speculative decoding, that may fail on standard dense models.

Our findings suggest a generalized training curriculum: progressively increasing a model's effective capacity throughout training yields superior results. While this work focused on increasing effective depth via layer dropout, this "model growing" strategy can be extended to other dimensions—such as model width—and granularities—such as quantization bit-widths, or unstructured sparsity—using similar spatial distributions and temporal schedules. We envision this framework as a foundational pillar for efficient large-scale pre-training, encouraging the re-adoption of layer dropout as a primary enabler for flexible model growth.

Future work can also include deducing the optimal maximum dropout rates for specific model scales and data budgets to maximize the Pareto frontier of training and inference efficiency, as well as investigating "learned" depth-aware training mechanisms, where the model dynamically identifies optimal skipping paths rather than relying on stochastic selection.

Impact Statement

This paper presents work whose goal is to advance the field of machine learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

Acknowledgments

We would like to thank Shaheer Mohammad and Sam McPhail for infrastructure support at Cerebras.

References

Bilge Acun, Benjamin Lee, Fiodar Kazhamiaka, Kiwan Maeng, Manoj Chakkaravarthy, Udit Gupta, David Brooks, and Carole-Jean Wu. Carbon explorer: A holistic approach for designing carbon aware datacenters. *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2023.

Naman Agarwal, Siddhartha R. Dalal, and Vishal Misra. Gradient dynamics of attention: How cross-entropy sculpts bayesian manifolds, 2026. URL https://arxiv.org/abs/2512.22473.

Shane Bergsma, Nolan Simran Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, and Joel Hestness. Power lines: Scaling laws for weight decay and batch size in LLM pre-training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a. URL https://openreview.net/forum?id=bFXbLQzRoZ.

Shane Bergsma, Bin Claire Zhang, Nolan Dey, Shaheer Muhammad, Gurpreet Gosal, and Joel Hestness. Scaling with collapse: Efficient and predictable training of LLM families, 2025b. URL https://arxiv.org/abs/2509.25087.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HylxE1HKwS.

Ruisi Cai, Saurav Muralidharan, Greg Heinrich, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. Flextron: Many-in-one flexible large language model. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=9vKRhnflAs.

Ruisi Cai, Saurav Muralidharan, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. LLaMaFlex: Many-in-one LLMs via generalized pruning and weight sharing. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=AyC4uxx2HW.

Shaofeng Cai, Yao Shu, and Wei Wang. Dynamic routing networks. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 3588–3597, January 2021.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. PaLM: Scaling language modeling with pathways, 2022. URL https://arxiv.org/abs/2204.02311.

Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. Analysis of DAWNBench, a time-to-accuracy machine learning performance benchmark. *ACM SIGOPS Operating Systems Review*, 53(1):14–25, 2019.

Francesco D'Angelo, Maksym Andriushchenko, Aditya Varre, and Nicolas Flammarion. Why do we need weight decay in modern deep learning? In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=YrAxxscKM2.

Fnu Devvrit, Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit S Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham M. Kakade, Ali Farhadi, and Prateek Jain. MatFormer: Nested transformer for elastic inference. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=fYa6ezMxD5.

Nolan Dey, Quentin Anthony, and Joel Hestness. The practitioner's guide to the maximal update parameterization, 2024. URL https://www.cerebras.ai/blog/the-practitioners-guide-to-the-maximal-update-parameterization.

Nolan Simran Dey, Bin Claire Zhang, Lorenzo Noci, Mufan Li, Blake Bordelon, Shane Bergsma, Cengiz Pehlevan, Boris Hanin, and Joel Hestness. Don't be lazy: CompleteP enables compute-efficient deep transformers. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=lMU2kaMANl.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. LayerSkip: Enabling early exit inference and self-speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.681. URL https://aclanthology.org/2024.acl-long.681/.

Sara Elkerdawy, Mostafa Elhoushi, Abhineet Singh, Hong Zhang, and Nilanjan Ray. To filter prune, or to layer prune, that is the question. In Hiroshi Ishikawa, Cheng-Lin Liu, Tomas Pajdla, and Jianbo Shi (eds.), *Computer Vision – ACCV 2020*, pp. 737–753, Cham, 2021. Springer International Publishing. ISBN 978-3-030-69535-4.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SylO2yStDr.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically, 2017. URL https://arxiv.org/abs/1712.00409.

Dylan Hillier, Leon Guertler, Bobby Cheng, and Cheston Tan. STLM engineering report: Dropout, 2024. URL https://arxiv.org/abs/2409.05423.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012. URL https://arxiv.org/abs/1207.0580.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (eds.), *Computer Vision – ECCV 2016*, pp. 646–661, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46493-0.

Benyamin Jamialahmadi, Parsa Kavehzadeh, Mehdi Rezagholizadeh, Parsa Farinneya, Hossein Rajabzadeh, Aref Jafari, Boxing Chen, and Marzieh S. Tahaei. Balcony: A lightweight approach to dynamic inference of generative language models. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 24853–24867, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1263. URL https://aclanthology.org/2025.emnlp-main.1263/.

Yikun Jiang, Huanyu Wang, Lei Xie, Hanbin Zhao, Chao Zhang, Hui Qian, and John C.S. Lui. D-LLM: A token adaptive computing resource allocation strategy for large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=UIOjGTKHQG.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Tanishq Kumar, Zachary Ankner, Benjamin F. Spector, Blake Bordelon, Niklas Muennighoff, Mansheej Paul, Cengiz Pehlevan, Christopher Ré, and Aditi Raghunathan. Scaling laws for precision, 2024. URL https://arxiv.org/abs/2411.04330.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Duyu Tang, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding for accelerating LLMs via double early exiting. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=lT3oc04mDp.

Houjun Liu, John Bauer, and Christopher D Manning. Drop dropout on single epoch language model pretraining. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 2157–2166, Vienna, Austria,

July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.111. URL https://aclanthology.org/2025.findings-acl.111/.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Yao Lu, Hao Cheng, Yujie Fang, Zeyu Wang, Jiaheng Wei, Dongwei Xu, Qi Xuan, Xiaoniu Yang, and Zhaowei Zhu. Reassessing layer pruning in LLMs: New insights and methods. *arXiv preprint arXiv:2411.15558*, 2024.

Qingkai Meng, Hao Zheng, Zhenhui Zhang, ChonLam Lao, Chengyuan Huang, Baojia Li, Ziyuan Zhu, Hao Lu, Weizhen Dang, Zitong Lin, et al. Astral: A datacenter infrastructure for large language model training at scale. In *Proceedings of the ACM SIGCOMM 2025 Conference*, pp. 609–625, 2025.

Reza Moradi, Reza Berangi, and Behrouz Minaei. A survey of regularization strategies for deep models. *Artificial Intelligence Review*, 53(6):3947–3986, 2020.

Pushparaja Murugan and Shanmugasundaram Durairaj. Regularization and optimization strategies in deep convolutional neural network. *arXiv preprint arXiv:1712.04711*, 2017.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1797–1807, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1206. URL https://aclanthology.org/D18-1206/.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on GPU clusters using Megatron-LM. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 1–15, 2021.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel HAZIZA, Francisco Massa, Alaaeldin El-Nouby, et al. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=a68SUt6zFt. Featured Certification.

Ofir Press, Noah Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=R8sQPpGCv0.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. URL https://api.semanticscholar.org/CorpusID:160025533.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models, 2024. URL https://arxiv.org/abs/2404.02258.

Sebastian Raschka. From GPT-2 to GPT-OSS: Analyzing the architectural advances, Aug 2025. URL https://magazine.sebastianraschka.com/p/from-gpt-2-to-gpt-oss-analyzing-the.

Imrus Salehin and Dae-Ki Kang. A review on dropout regularization approaches for deep neural networks within the scholarly domain. *Electronics*, 12(14), 2023. ISSN 2079-9292. doi: 10.3390/electronics12143106. URL https://www.mdpi.com/2079-9292/12/14/3106.

Mohammad Samragh, Seyed Iman Mirzadeh, Keivan Alizadeh-Vahid, Fartash Faghri, Minsik Cho, Moin Nabi, Devang Naik, and Mehrdad Farajtabar. Scaling smart: Accelerating large language model pre-training with small model initialization. In Mehdi Rezagholizadeh, Peyman Passban, Soheila Samiee, Vahid Partovi Nia, Yu Cheng, Yue Deng, Qun Liu, and Boxing Chen (eds.), *Proceedings of The 4th NeurIPS Efficient Natural*

Language and Speech Processing Workshop, volume 262 of *Proceedings of Machine Learning Research*, pp. 1–13. PMLR, 14 Dec 2024. URL https://proceedings.mlr.press/v262/samragh24a.html.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=uLYc4L3C81A.

Li Shen, Yan Sun, Zhiyuan Yu, Liang Ding, Xinmei Tian, and Dacheng Tao. On efficient training of large-scale deep learning models. *ACM Computing Surveys*, 57(3):1–36, 2024.

Pierre Stock, Angela Fan, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme model compression. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=dV19Yyi1fS3.

Ali Taghibakhshi, Sharath Turuvekere Sreenivas, Saurav Muralidharan, Ruisi Cai, Marcin Chochowski, Ameya Sunil Mahabaleshwarkar, Yoshi Suhara, Oluwatobi Olabiyi, Daniel Korzekwa, Mostofa Patwary, et al. Nemotron elastic: Towards efficient many-in-one reasoning LLMs, 2025. URL https://arxiv.org/abs/2511.16664.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using DropConnect. In Sanjoy Dasgupta and David McAllester (eds.), *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pp. 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL https://proceedings.mlr.press/v28/wan13.html.

Irene Wang, Newsha Ardalani, Mostafa Elhoushi, Daniel Jiang, Samuel Hsia, Ekin Sumbul, Divya Mahajan, Carole-Jean Wu, and Bilge Acun. CATransformers: Carbon aware transformers through joint model-hardware optimization, 2025. URL https://arxiv.org/abs/2505.01386.

Sida Wang and Christopher Manning. Fast dropout training. In *international conference on machine learning*, pp. 118–126. PMLR, 2013.

Xin Wang, Yudong Chen, and Wenwu Zhu. A survey on curriculum learning, 2021. URL https://arxiv.org/abs/2010.13166.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared LLaMA: Accelerating language model pre-training via structured pruning. 2024. URL https://openreview.net/forum?id=09iOdaeOzp.

Fuzhao Xue, Yao Fu, Wangchunshu Zhou, Zangwei Zheng, and Yang You. To repeat or not to repeat: Insights from scaling LLM under token-crisis, 2023. URL https://arxiv.org/abs/2305.13230.

Greg Yang and Edward J Hu. Tensor programs IV: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, pp. 11727–11737. PMLR, 2021.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282, Bangkok, Thailand, August 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.607. URL https://aclanthology.org/2024.acl-long.607/.

Minjia Zhang and Yuxiong He. Accelerating training of transformer-based language models with progressive layer dropping. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 14011–14023. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/a1140a3d0df1c81e24ae954d935e8926-Paper.pdf.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. OPT: Open pre-trained transformer language models, 2022. URL https://arxiv.org/abs/2205.01068.

Zhengyan Zhang, Yixin Song, Guanghui Yu, Xu Han, Yankai Lin, Chaojun Xiao, Chenyang Song, Zhiyuan Liu, Zeyu Mi, and Maosong Sun. Relu-squared wins: Discovering efficient activation functions for sparse LLMs. *arXiv preprint arXiv:2402.03804*, 2024b.

Appendix

Glossary

Alternating Layer Distribution (ALD) A dropout distribution where dropout is not applied on the first layer, but is applied on alternating layers proceeding that. It is expressed mathematically as: $p^{\ell,t}_{\text{ALD}} = p_{\max} \cdot \mathbf{1}_{\ell \equiv 1 (\text{mod } 2)}$. 8

Constant Time Schedule A temporal schedule for layer dropout where the dropout rate remains constant throughout training. It is expressed mathematically as $p^{\ell,t}_{\text{const}} = p^{\ell}_{\text{dist}}$. 8

Decreasing Time Schedule (DTS) A temporal schedule for layer dropout where the dropout rate is at its maximum at the start of pre-training and decays linearly to zero by the final training step. It is expressed mathematically as $p^{\ell,t}_{\text{DTS}} = p^{\ell}_{\text{dist}} \cdot \left(1 - \frac{t}{T}\right)$. In this work, we demonstrate that this schedule helps stabilize early training while allowing the model to settle into a dense state for final convergence. 9

FFN Feed Forward Networks. 4

Increasing Layer Distribution (ILD) A layer dropout configuration where the dropout rate starts at 0 for the first layer, i.e., $p^{\ell=0} = 0$, and linearly increases across layers to a maximum dropout rate. It is mathematically expressed as: $p^{\ell,t}_{\text{ILD}} = \frac{\ell}{L-1} \cdot p_{\max}$. 8

Increasing Time Schedule (ITS) A temporal schedule for layer dropout where the dropout rate starts at zero and increases linearly to its maximum by the final training step. It is expressed mathematically as $p^{\ell,t}_{\text{ITS}} = p^{\ell}_{\text{dist}} \cdot \left(\frac{t}{T}\right)$. 9

Layer Dropout A form of dropout where entire layers of a neural network (e.g., transformer blocks) are randomly skipped during training. Unlike standard dropout, which zeroes out individual activations, layer dropout operates at the structural level, reducing the effective depth of the network on each forward pass. In the context of transformers, we use this term to indicate applying layer dropout on the granularity of a whole transformer block. 2, 6, 23

Stochastic Depth An alternative term for Layer Dropout. 2

Sub-Layer Dropout In the context of transformers, refers to applying layer dropout on attention and FFN blocks independently. 6

Uniform Distribution A layer dropout configuration where dropout rates of all layers are set to the same value, and can be mathematically expressed as $p^{\ell,t}_{\text{uniform}} = p_{\max}$. 7

A Experimental Settings

Model	Hidden Dim. D	Layers L	Heads	Head Size	FFN Mult.
271M	640	13	10	64	8
504M	896	17	14	64	8
906M	1152	23	9	128	8
1.8B	1536	30	12	128	8
3.9B	2048	40	16	128	8
8.2B	2688	52	21	128	8

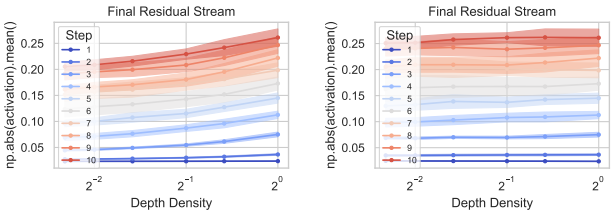
Table A.1: Model architectures used in the experiments.

Table A.2: Summary of SP, µP, and CompleteP with layer dropout rate for a transformer model. Terms related to width (introduced by μP Yang & Hu (2021)), depth (introduced by CompleteP Dey et al. (2025)), data size (introduced by Bergsma et al. (2025a)), and dropout (introduced in this paper) controls are highlighted in orange, green, brown, and purple respectively. Additional tunable parameters are highlighted in blue. *Hidden* refers to all linear layers in the transformer backbone. Layer density, ρ is the complement of layer dropout, p such that $\rho = 1 - p$.

Parameterization	Base Training	Scaled Training
Width	d_{base}	$d_{\text{base}} \cdot m_d$
Depth	L_{base}	$L_{\text{base}} \cdot m_L$
Dataset Size	D_{base}	$D_{\text{base}} \cdot m_D$
Layer Density	1	ρ
Model Size	$N_{\text{base}} = g(d_{\text{base}}, L_{\text{base}})$	$N = g(d_{\text{base}} \cdot m_d, L_{\text{base}} \cdot m_L)$
Tokens per Parameter	$\text{TPP}_{\text{base}} = D_{\text{base}}/N_{\text{base}}$	$\text{TPP} = D_{\text{base}} \cdot m_D/N$
Batch Size	B_{base}	$B_{\text{base}} \cdot m_D^{0.4}$
Timescale (AdamW)	$\tau\text{EMA}_{\text{base}}$	$\tau\text{EMA} = \tau\text{EMA}_{\text{base}} \cdot (\frac{\text{TPP}}{\text{TPP}_{\text{base}}})^{-0.5}$
Emb. Init. Var.	σ^2_{base}	σ^2_{base}
Emb. LR (AdamW)	η_{base}	η_{base}
Pre-LN Init. Var.	σ^2_{base}	σ^2_{base}
Pre-LN LR (AdamW)	η_{base}	η_{base}
Hidden Init. Var.	σ^2_{base}	$\sigma^2_{\text{base}} \cdot m_d^{-1}$
Hidden LR (AdamW)	η_{base}	$\eta_{\text{base}} \cdot m_d^{-1}$
Hidden Bias LR (AdamW)	η_{base}	η_{base}
Hidden WD (AdamW)	$\frac{B_{\text{base}}}{\eta_{\text{base}} \tau\text{EMA}_{\text{base}} D_{\text{base}}}$	$\frac{B_{\text{base}}}{\eta_{\text{base}} \tau\text{EMA} D_{\text{base}} m_D^{0.4}} \cdot m_d$
Attention Residual	$\mathbf{X}^l + f_{\text{attn}}(\mathbf{X}^l)$	$\mathbf{X}^l + {m_L}^{-1} \rho^{-1} \cdot f_{\text{attn}}(\mathbf{X}^l)$
FFN Residual	$\mathbf{Z}^l + f_{\text{ffn}}(\mathbf{Z}^l)$	$\mathbf{Z}^l + {m_L}^{-1} \rho^{-1} \cdot f_{\text{ffn}}(\mathbf{Z}^l)$
Final-LN Init. Var.	σ^2_{base}	σ^2_{base}
Final-LN LR (AdamW)	η_{base}	η_{base}
Unemb. Init. Var.	σ^2_{base}	σ^2_{base}
Unemb. LR (AdamW)	η_{base}	η_{base}
Unemb. Fwd.	$\mathbf{X}^L \mathbf{W}^\top_{\text{unemb}}$	$\mathbf{X}^L \mathbf{W}^\top_{\text{unemb}} \cdot m_d^{-1}$
AdamW ϵ (Residual blocks)	ϵ_{base}	$\epsilon_{\text{base}} \cdot m_d^{-1} \cdot {m_L}^{-1}$
AdamW ϵ (Emb. & Unemb.)	ϵ_{base}	$\epsilon_{\text{base}} \cdot m_d^{-1}$

B Coordinate Check

Fig. 2 and Fig. A.1 show coordinate check plots for uniform and non-uniform dropouts respectively. They show Frobenius norm of activations after merged residual streams from attention and FFN blocks across a 40 layer model after 10 training steps, using CompleteP (that scales residuals by $\frac{L_{\text{base}}}{L}$), r^ℓ_{train} for layer ℓ.



(a) $r^{l,t}_{\text{train}} = 1$ (b) $r^{l,t}_{\text{train}} = 1/\rho^{l,t}$ 👑

Figure A.1: Coordinate Check passing for non-uniform distribution.

C Additional Results

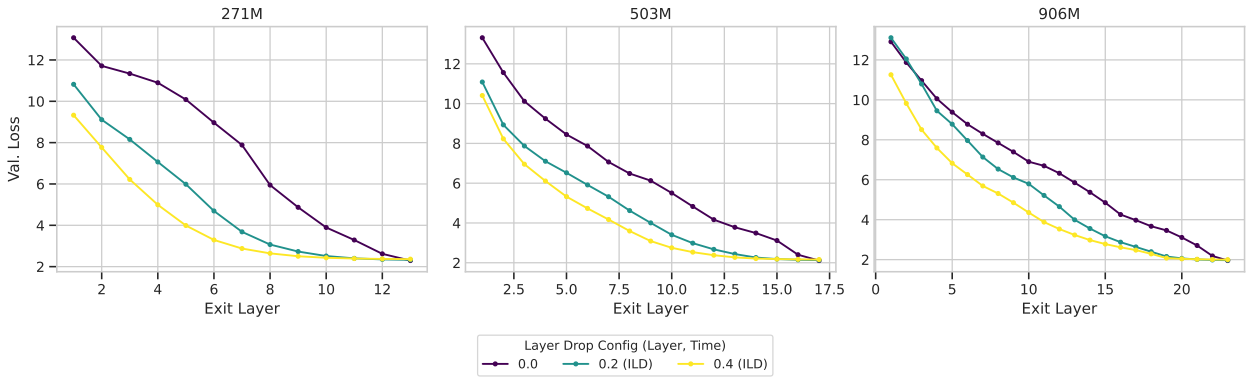
C.1 Downstream Tasks

Table A.3: Downstream task performance benchmarks.

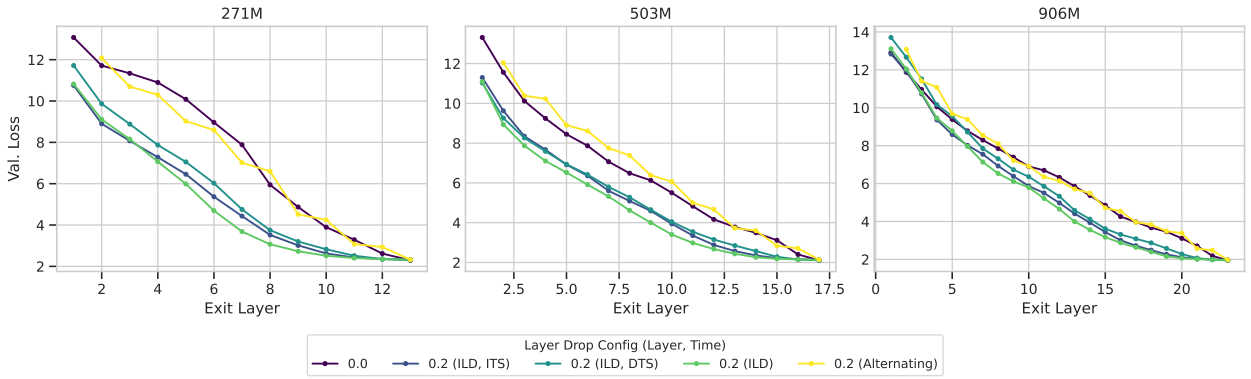
		Training Config.		Downstream Tasks ↑										
Size	**TPP**	**Layer Dropout**	**FLOPs Sav. ↑**	**BBH**	**PIQA**	**SIQA**	**Hella.**	**Wino.**	**ARC-c**	**ARC-e**	**OBQA**	**Lamb.**	**COPA**	**RACE**
271M	20	0	0%	0.083	0.593	0.358	0.293	0.500	0.224	0.415	0.274	0.226	0.580	0.265
271M	20	0.2 (ILD)	10%	0.095	0.579	0.349	0.285	0.505	0.224	0.399	0.262	0.218	0.580	0.273
271M	20	0.4 (ILD, DTS)	10%	0.106	0.584	0.358	0.290	0.523	0.241	0.409	0.274	0.229	0.600	0.269
271M	20	0.4 (ILD)	20%	0.128	0.584	0.345	0.283	0.499	0.234	0.400	0.266	0.223	0.610	0.253
503M	20	0	0%	0.178	0.594	0.360	0.316	0.515	0.246	0.441	0.282	0.280	0.560	0.279
503M	20	0.2 (ILD)	10%	0.198	0.592	0.357	0.306	0.497	0.234	0.446	0.272	0.290	0.620	0.266
503M	20	0.4 (ILD, DTS)	10%	0.162	0.586	0.361	0.309	0.516	0.239	0.443	0.274	0.281	0.670	0.268
503M	20	0.4 (ILD)	20%	0.167	0.596	0.358	0.301	0.499	0.241	0.434	0.270	0.283	0.640	0.263
906M	20	0	0%	0.206	0.611	0.360	0.353	0.493	0.242	0.493	0.286	0.337	0.660	0.286
906M	20	0.2 (ILD)	10%	0.220	0.607	0.386	0.343	0.510	0.242	0.477	0.288	0.339	0.670	0.300
906M	20	0.4 (ILD, DTS)	10%	0.226	0.609	0.365	0.353	0.527	0.257	0.484	0.272	0.345	0.680	0.305
906M	20	0.4 (ILD)	20%	0.221	0.613	0.357	0.334	0.513	0.245	0.471	0.274	0.330	0.650	0.292
1.9B	20	0	0%	0.251	0.640	0.385	0.404	0.528	0.282	0.543	0.290	0.398	0.670	0.313
1.9B	20	0.6 (ILD, DTS)	15%	0.241	0.636	0.376	0.400	0.5241	0.282	0.533	0.282	0.389	0.670	0.299
3.9B	20	0	0%	0.272	0.666	0.399	0.473	0.555	0.312	0.597	0.356	0.479	0.650	0.325
3.9B	20	0.8 (ILD, DTS)	20%	0.270	0.668	0.402	0.462	0.563	0.317	0.585	0.324	0.480	0.680	0.329
8.2B	20	0.99 (ILD, DTS)	25%	0.291	0.699	0.407	0.523	0.566	0.360	0.634	0.356	0.529	0.700	0.356

C.2 Zero-Shot Inference Benefits

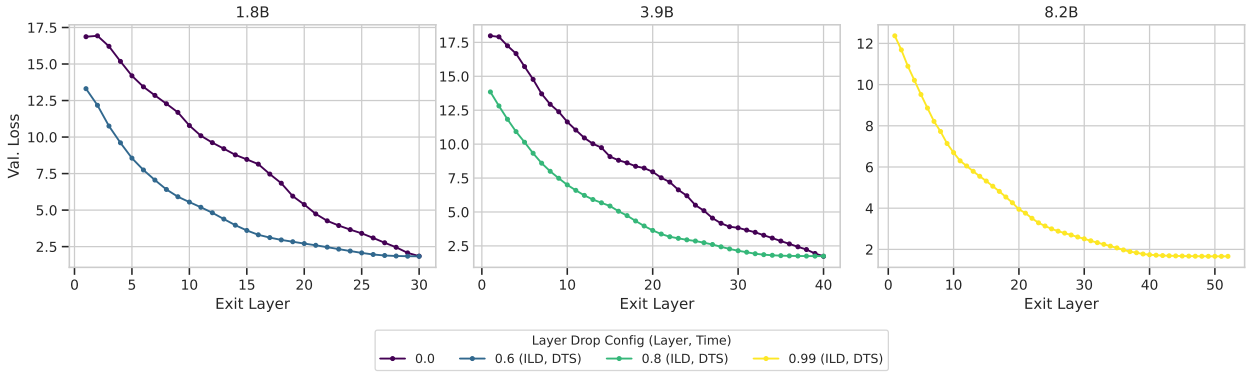
C.2.1 Early Exit



(a) Ablating different maximum dropout values for Linear Distribution.



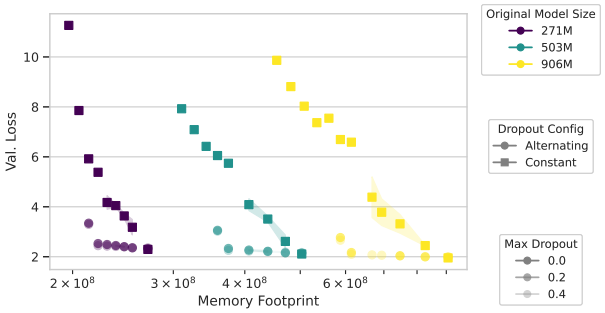
(b) Ablating different dropout configurations for the same maximum dropout of 0.2.



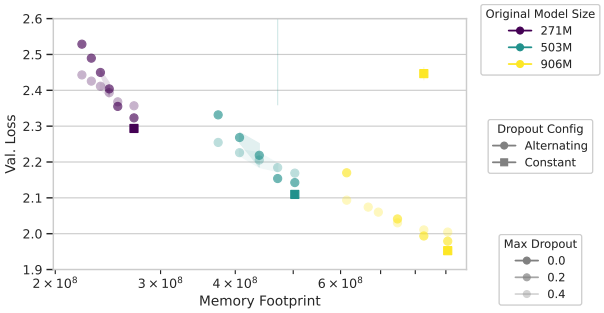
(c) Larger model sizes with aggressive maximum dropout rates.

Figure A.2: Comparison of early-exit validation losses for models trained with different dropout configurations. All models trained with 20 TPP.

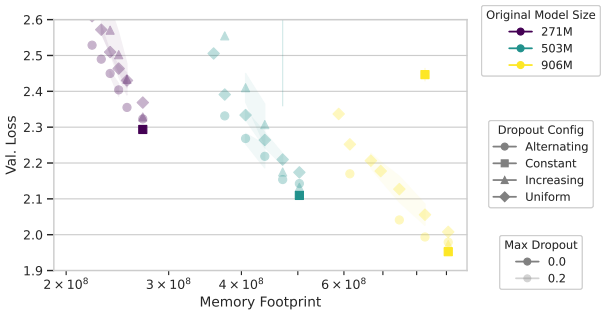
C.2.2 Intermediate Layer Skipping



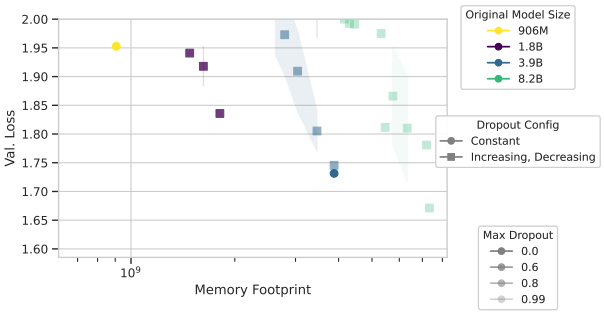
(a) Ablating different maximum dropout values for Alternate Distribution.



(b) Ablating different maximum dropout values for Alternate Distribution, zooming on skipping configurations with lower losses.



(c) Ablating different dropout configurations for the same maximum dropout of 0.2.



(d) Larger model sizes with aggressive maximum dropout rates.

Figure A.3: Comparison of intermediate layer skipping validation losses for models trained with different dropout configurations. All models trained with 20 TPP.