PREFILL DOES NOT NEED THE FULL FFN

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent studies have shown that computation during prefill can be reduced, with several approaches focusing on skipping entire Transformer layers. In this work, we investigate prefill computation at a finer granularity by reducing feed-forward network (FFN) capacity. We ask two questions: **(Q1)** To what extent can FFN capacity be reduced during prefill while preserving model quality? **(Q2)** How should the retained FFN capacity be distributed across layers? To answer these questions, we introduce a *Nested FFN* architecture, in which prefill activates only a fixed subset of intermediate FFN channels while decode uses the full FFN, with both phases sharing the same parameters. Our experiments show that substantial FFN capacity (37.5% in multiple settings) can be removed during prefill, including in early Transformer layers.

1 INTRODUCTION

Large language model inference consists of two functionally distinct phases. Prefill primarily constructs contextual representations for future tokens, whereas decode additionally performs autoregressive next-token prediction. Despite these different roles, standard Transformers (Vaswani et al., 2017) use the same network architecture and computation in both phases.

Recent work has shown that this symmetry is not necessary. POP (He et al., 2026) demonstrates that deeper Transformer layers can be skipped during prefill while remaining important for decoding, while YOCO (Sun et al., 2024) and CED (DeepSeek-AI, 2026) redesign the architecture to avoid substantial later-layer computation during prefill. Together, these results demonstrate that computation essential for decode may be unnecessary for prefill. However, they characterize this computational redundancy primarily at the granularity of entire layers. This motivates a finer-grained question: How much computation is needed within each Transformer layer during prefill to preserve model quality?

We study this question using FFN width as a simple and controlled mechanism for adjusting the computational capacity allocated to prefill within each Transformer layer. Specifically, we ask: **(Q1)** To what extent can FFN capacity be reduced during prefill while preserving model quality? **(Q2)** How should the retained FFN capacity be distributed across layers? To answer these questions, we develop a *Nested FFN* architecture, in which prefill executes only a subset of intermediate channels while decoding retains the full FFN, with both phases sharing the same parameters. We instantiate Nested FFN in two configurations. The first applies a uniform reduction in FFN capacity across all layers, providing a controlled measure of how much overall capacity prefill requires. The second allows FFN capacity to be allocated non-uniformly across layers, enabling us to investigate whether some layers require more capacity than others under a fixed computational budget. Both configurations are learned during training and fixed at inference time, requiring no input-dependent routing or dynamic execution decisions.

Our first finding is that a substantial fraction of FFN capacity can be removed during prefill while largely preserving model quality. As the uniform prefill width is progressively reduced, performance remains stable, with noticeable degradation emerging only under tighter budgets. These results characterize the quality–capacity trade-off of FFN reduction during prefill and reveal substantial redundancy within the FFN computation.

We next study how the retained FFN capacity should be distributed across layers by allowing a non-uniform, layer-adaptive allocation. Figure 1 shows a clear trend across the evaluated models: earlier layers generally retain more capacity, yet their learned widths remain below the full FFN. This

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

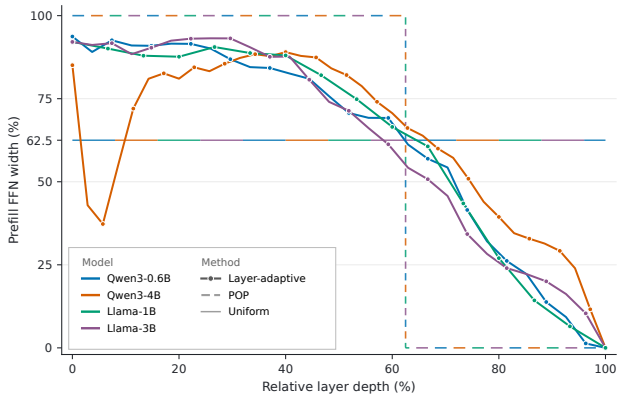

Figure 1: Learned layer-adaptive FFN allocations during prefill across different LLMs, with a retained FFN width of 62.5% (i.e., a 37.5% reduction) while preserving model quality. Uniform allocation and POP, which completely skips approximately 37.5% of layers, are included as reference profiles. Despite model-specific variations, the learned allocations consistently exhibit non-uniform, graded profiles across the evaluated models.

reveals that removable prefill computation exists within individual Transformer layers, including early layers, rather than being limited to the complete removal of later layers. Under matched prefill budgets, the importance of how FFN capacity is allocated across layers becomes increasingly pronounced as the available capacity decreases.

Our main contributions are summarized as follows:

- We develop a Nested FFN architecture for reduced FFN computation in prefill, a shared-parameter mechanism that executes a fixed subset of FFN channels during prefill while retaining the full FFN for decode.
- Using uniform FFN reduction, we characterize how model quality changes as FFN capacity is progressively reduced, showing that substantial FFN capacity can be reduced while largely preserving model quality.
- Using layer-adaptive FFN reduction, we show that FFN capacity can be further reduced without compromising model quality. This finding reveals additional opportunities for reducing prefill computation beyond existing architectures such as YOCO (Sun et al., 2024), which primarily eliminate computation at the granularity of entire layers.

In contrast to existing work, we focus on quantitatively reducing prefill computation while preserving model quality.

2 Method

2.1 Nested FFN

We consider a decoder-only Transformer with L layers, hidden dimension d, and FFN intermediate width d_{ff}. For a hidden state $h \in \mathbb{R}^d$, the FFN at layer l is

$$\text{FFN}_l(h) = W_{\text{down}}^{(l)}\, \sigma\Big(W_{\text{up}}^{(l)} h\Big), \qquad W_{\text{up}}^{(l)} \in \mathbb{R}^{d_{\text{ff}} \times d}, \quad W_{\text{down}}^{(l)} \in \mathbb{R}^{d \times d_{\text{ff}}}, \tag{1}$$

where σ denotes the activation function. Gated variants such as SwiGLU (Shazeer, 2020) introduce an additional projection with the same intermediate dimension; all width reductions below are applied to the corresponding channels consistently.

During inference-time prefill, a reduced FFN with width ratio $\rho \in (0, 1]$ executes a subset of intermediate channels arranged as a contiguous prefix within the shared FFN:

$$\text{FFN}_l^{\rho}(h) = W_{\text{down}}^{(l)}[:, :\rho d_{\text{ff}}]\, \sigma\Big(W_{\text{up}}^{(l)}[:\rho d_{\text{ff}}, :]\, h\Big). \tag{2}$$

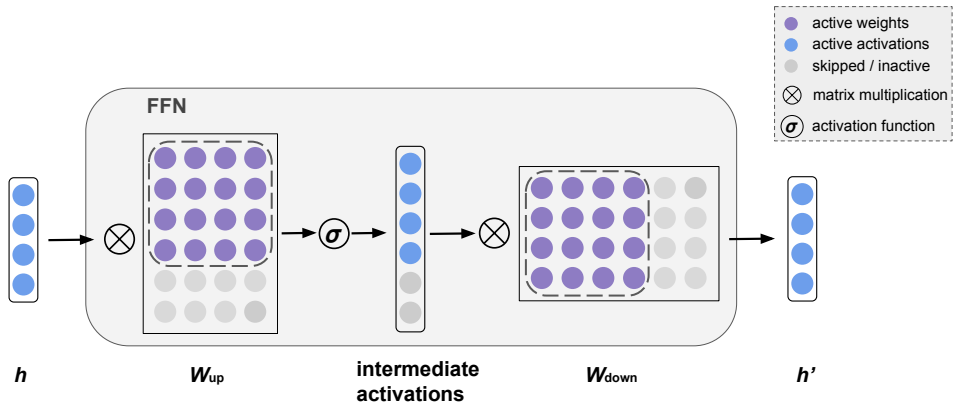

Figure 2: Overview of Nested FFN at inference. The reduced FFN executes a fixed subset of intermediate channels, computing only the corresponding activations and weight slices while sharing parameters with the full FFN.

The retained width directly controls the amount of dense FFN computation. Since the intermediate dimension of the FFN projections is reduced from d_{ff} to ρd_{ff}, the projection computation for these prompt positions scales linearly with ρ. For example, retaining 62.5% of the FFN width removes 37.5% of the corresponding FFN projection computation.

As illustrated in Figure 2, the reduced FFN computes only the selected intermediate activations and their corresponding weight slices, while the full FFN retains all channels. The two computations share the same parameters, introducing no additional model parameters. For gated FFNs, the same intermediate-channel subset is applied consistently to the up, gate, and down projections. Because the selected channels form a contiguous prefix at inference time, the reduced FFN can be executed with standard dense matrix multiplications.

We further reserve a suffix window of size ω in the prompt that uses the full FFN. For a prompt of length P, the first $P - \omega$ positions use the reduced FFN, whereas the final ω prompt positions and all generated tokens use the full-width FFN.

2.2 Prefill Width Allocation

Figure 2 illustrates the inference-time structure of Nested FFN. We next introduce two width-allocation strategies that determine the FFN subset used for prefill at each layer.

Uniform Width Reduction. The simplest allocation uses the same reduced width at every layer:

$$\rho^{(l)} = \rho, \qquad l = 1, \ldots, L. \tag{3}$$

For Uniform Width Reduction, the first ρd_{ff} channels are selected as the FFN subset for prefill, while the full FFN is used for decode. We further fine-tune the FFN parameters under this reduced prefill computation configuration. Sweeping ρ provides a controlled way to vary the amount of FFN computation used during prefill and characterize how model quality changes as the available capacity is reduced.

Layer-adaptive Width Reduction. To study how the retained capacity should be distributed across layers, we allow the width ratio $\rho^{(l)}$ to vary across layers while constraining its average.

For each layer l, we introduce learnable channel logits $z^{(l)} \in \mathbb{R}^{d_{\text{ff}}}$ and define a binary mask

$$m^{(l)} = \mathbb{I}\left[z^{(l)} > 0\right], \tag{4}$$

which indicates that the intermediate channels will be used during prefill. The mask is optimized using a straight-through estimator (STE) (Bengio et al., 2013).

Let $\mathcal{L}_{\mathrm{LM}}$ denote the standard next-token cross-entropy loss. We define the layer-wise width ratio and the global budget objective as

$$\rho^{(l)} = \frac{\|m^{(l)}\|_0}{d_{\mathrm{ff}}}, \qquad \mathcal{L}_{\mathrm{gated}} = \mathcal{L}_{\mathrm{LM}} + \lambda \left(\frac{1}{L} \sum_{l=1}^{L} \rho^{(l)} - \rho_{\mathrm{target}} \right)^2, \tag{5}$$

where λ controls the strength of the budget penalty and ρ_{target} denotes the target average prefill width ratio. The individual layer widths are learned under a shared average-width budget rather than specified in advance. Since the budget is enforced through a penalty, the realized mean width may differ slightly from ρ_{target}.

During training, the learned active channels may be scattered. After training, we permute the FFN intermediate channels at each layer so that those selected by $m^{(l)}$ occupy a contiguous prefix. The same permutation is applied consistently to W_{up}, W_{down}, and the corresponding projection in gated FFNs. This preserves the full-width FFN function while converting the learned mask into a standard dense prefix slice with layer-specific width $\rho^{(l)}$.

2.3 Training

We start from an instruction-tuned checkpoint and fine-tune the FFN parameters so that the same model can use reduced FFN computation during prefill while retaining the full FFN for the prompt suffix window and decode. We use generic long-context text for fine-tuning, avoiding dependence on a particular instruction-data mixture while exposing the model to the long-context regime studied in our experiments.

A standard causal language-modeling sequence does not provide an explicit prompt boundary separating prefill from decode. Therefore, for each selected query position t, we sample a distance δ_t from a predefined range, representing how far t lies after a hypothetical prompt boundary. With this boundary, the final ω prompt positions and decoded positions use the full FFN, while earlier prompt positions use the reduced FFN. We refer to the key and value states derived from hidden representations computed with the full and reduced FFNs as *full K/V* and *reduced K/V*, respectively. For each preceding position $j \le t$, query t uses

$$\begin{cases} \text{full K/V of } j, & \text{if } t - j \le \delta_t + \omega, \\ \text{reduced K/V of } j, & \text{otherwise.} \end{cases}$$

Since multiple query positions are sampled from one sequence, the same position j may provide full K/V for one query and reduced K/V for another. We therefore compute both full and reduced hidden representations for every token position at each Transformer layer and project the corresponding K/V states. All model parameters are shared between the two representations.

We optimize the standard next-token cross-entropy loss $\mathcal{L}_{\mathrm{LM}}$ on the selected query positions. Only the FFN parameters are fine-tuned, while the remaining model parameters are frozen. For Layer-adaptive Width Reduction, the width allocation is optimized jointly using the budget-regularized objective in Eq. 5.

2.4 Inference

At inference time, the prompt boundary is known and only a single residual stream is required. For a prompt of length P, the first $P - \omega$ prompt positions use reduced-width FFNs, while the final ω prompt positions and all generated positions use the full-width FFN.

For Uniform Width Reduction, every layer uses the same reduced ratio ρ. For Layer-adaptive Width Reduction, layer l uses its learned ratio $\rho^{(l)}$. After the channel permutation described above, both configurations are implemented using contiguous prefix slices and standard dense GEMMs.

3 EXPERIMENTS

3.1 EXPERIMENTAL SETUP

Models. We use instruction-tuned variants throughout downstream evaluation. We consider Qwen3-0.6B and Qwen3-4B (Yang et al., 2025), as well as Llama-3.2-1B-Instruct and Llama-3.2-3B-Instruct (Grattafiori et al., 2024), covering two model families and multiple model scales.

Compared configurations. We compare four prefill configurations. *Vanilla* is obtained by adapting the official checkpoint on our training data, controlling for the effect of continued training. Nested FFN has two instantiations: *Uniform Width Reduction*, which uses the same reduced FFN width at every layer, and *Layer-adaptive Width Reduction*, which learns how to distribute a fixed average prefill width across layers (Sec. 2.2). We additionally compare with *POP* (He et al., 2026), which uses a binary depth-wise allocation during prefill, completely skipping selected later Transformer layers while retaining the remaining layers. For a fair FFN-level comparison with POP, we retain attention computation and fully skip only the selected later-layer FFNs. In addition, both Nested FFN configurations and the controlled POP baseline use a full-width suffix window of $\omega = 512$, while decode always uses the full FFN. We verify in Appendix A that the controlled POP variant used in our experiments remains competitive with Original POP.

Training. We fine-tune Vanilla, Uniform Width Reduction, and Layer-adaptive Width Reduction on PG19 (Rae et al., 2019), comprising approximately 1.05B tokens from $8{,}425$ documents. Training sequences are contiguous 32K-token spans from individual documents, without cross-document packing. All configurations use the same data, data order, and random seed. We fine-tune for $2{,}000$ steps using AdamW (Loshchilov & Hutter, 2017) with learning rate 5×10^{-5}, zero weight decay, global batch size 8, and 20 warmup steps followed by cosine decay to 5×10^{-6}. Only the FFN parameters are updated; all other model parameters are frozen. For Uniform Width Reduction and Layer-adaptive Width Reduction, we fix the suffix-window size to $\omega = 512$ and sample the query-boundary distance δ_t uniformly from $[0, 512]$ during fine-tuning (Sec. 2.3). For Layer-adaptive Width Reduction, we set the budget-penalty coefficient to $\lambda = 8$ for all experiments. POP is applied to the fine-tuned Vanilla checkpoint.

Evaluation. We mainly evaluate long-context capability, where reducing prefill computation is particularly valuable as the computational cost grows with prompt length and preserving performance remains challenging. We use RULER (Hsieh et al., 2024) to evaluate retrieval, aggregation, and reasoning over long contexts, and LongBench (Bai et al., 2024) to evaluate performance on a broader set of real-world long-context tasks such as question answering and summarization. For RULER, we evaluate context lengths of 8K, 16K, and 32K and report the average score across tasks at each length. For LongBench, we evaluate all English tasks, report their average score, and follow the official model-specific context limits and middle-truncation protocol.

We additionally evaluate general capabilities to verify that reduced-width prefill does not degrade standard downstream performance. Specifically, MMLU (Hendrycks et al., 2020) evaluates broad knowledge and reasoning, GSM8K (Cobbe et al., 2021) mathematical reasoning, ARC-Challenge (Clark et al., 2018) science question answering, and HellaSwag (Zellers et al., 2019) commonsense reasoning. We use 5-shot, 5-shot, 25-shot, and 10-shot prompting, respectively. Following the official evaluation protocols, we apply the corresponding chat template for each model. We evaluate Qwen3 in non-thinking mode. Since reduced-width prefill is applied only before the full-width suffix window, we verify that most general-capability prompts extend beyond $\omega = 512$ under both Qwen and Llama tokenizers; detailed prompt-length statistics are provided in Appendix B.

Implementation and hardware. Our implementation is based on PyTorch (Paszke et al., 2019) and Hugging Face Transformers (Wolf et al., 2020). Training is conducted using eight NVIDIA A100 80 GB GPUs and four NVIDIA GH200 GPUs with bfloat16 precision.

3.2 Q1: TO WHAT EXTENT CAN FFN WIDTH BE REDUCED DURING PREFILL?

We characterize how model quality changes as the prefill FFN width is reduced across layers while keeping decode at full width. Table 1 reports results across several width settings on Qwen3-0.6B.

Table 1: Performance across prefill FFN width budgets on Qwen3-0.6B. Uniform Width Reduction uses the same width ratio at every Transformer layer, while Layer-adaptive Width Reduction learns a different width for each layer under the same average budget. Decode always uses the full FFN width. **Rel.** reports the average performance retention relative to Vanilla across RULER 8K, 16K, 32K, and LongBench. Best results within each matched width budget are shown in **bold**.

Method	Avg. Prefill Width	RULER 8K	RULER 16K	RULER 32K	LongBench	Rel.
Vanilla	100%	68.75	64.06	58.03	32.09	100.0%
Uniform Reduction	87.5%	**72.10**	**66.96**	**60.56**	31.24	102.8%
Layer-adaptive Reduction	$\sim$ 87.5%	70.18	66.73	60.40	**32.68**	**103.0%**
Uniform Reduction	75%	68.95	65.09	**58.61**	30.73	99.7%
Layer-adaptive Reduction	$\sim$ 75%	**69.64**	**65.49**	58.29	**32.89**	**101.6%**
Uniform Reduction	62.5%	65.61	62.07	55.91	30.36	95.8%
Layer-adaptive Reduction	$\sim$ 62.5%	**70.02**	**66.82**	**61.84**	**31.40**	**102.6%**
Uniform Reduction	50%	63.35	58.80	53.62	29.05	91.7%
Layer-adaptive Reduction	$\sim$ 50%	**68.21**	**63.61**	**57.79**	**30.55**	**98.3%**

Focusing on Uniform Width Reduction, at 75% prefill width or above, performance remains effectively at the Vanilla level on average. As the available prefill width decreases further, performance degrades gradually. Overall, these results show that a substantial fraction of FFN capacity can be removed during prefill while preserving strong model quality, providing a quantitative view of the extent of removable prefill computation.

3.3 Q2: How Should the Retained FFN Capacity Be Distributed Across Layers?

Q1 shows that substantial FFN capacity can be removed during prefill while preserving model quality. We next ask how the retained capacity should be distributed across layers. To study this, we fix the average prefill FFN width and allow the available capacity to be distributed non-uniformly across Transformer layers.

Learned depth structure. The learned allocation exhibits a clear depth-dependent and graded structure. Earlier layers generally retain more FFN width, but importantly, their learned widths also remain below the full FFN. The allocation generally decreases toward deeper layers. As shown in Figure 1, this pattern appears across the evaluated models at a retained width of 62.5%, while Layer-adaptive Width Reduction maintains Vanilla-level performance across these models (Table 2).

Does layer-wise allocation matter? We next compare Layer-adaptive Width Reduction with Uniform Width Reduction under the same average prefill width. Table 1 shows that the importance of layer-wise allocation becomes more pronounced as the available capacity decreases. At relatively large widths, both allocations remain close to Vanilla. As the width is reduced further, however, Layer-adaptive Width Reduction preserves substantially more performance than distributing the same capacity uniformly across layers. These results show that the tolerable extent of prefill FFN reduction depends not only on the total retained capacity, but also on how that capacity is distributed across depth.

3.4 Cross-model Validation

We next test whether the findings from Q1 and Q2 persist across additional model scales and families. We evaluate all models at an average retained prefill FFN width of 62.5%, corresponding to a 37.5% reduction, and additionally consider a tighter 50% retained-width budget for Qwen3-4B and Llama-3.2-3B.

Table 2: Cross-model performance under reduced-width prefill. All models are evaluated at an average prefill FFN budget of approximately 62.5%, with additional 50% evaluations on Qwen3-4B and Llama-3.2-3B. **Rel.** reports the average performance retention relative to the corresponding Vanilla model across RULER 8K, 16K, 32K, and LongBench. The highest **Rel.** among reduced-width methods within each matched model–budget setting is shown in **bold**.

Model	Method	Avg. Prefill Width	RULER 8K	RULER 16K	RULER 32K	LongBench	Rel.
Qwen3-0.6B	Vanilla	100%	68.75	64.06	58.03	32.09	100.0%
	POP	$\sim 62.5\%$	54.19	49.10	40.86	29.01	79.1%
	Uniform	62.5%	65.61	62.07	55.91	30.36	95.8%
	Layer-adaptive	$\sim 62.5\%$	70.02	66.82	61.84	31.40	**102.6%**
Qwen3-4B	Vanilla	100%	91.79	89.85	86.15	48.57	100.0%
	POP	$\sim 62.5\%$	89.24	86.51	82.66	47.71	96.9%
	Uniform	62.5%	91.55	89.88	85.83	47.38	99.2%
	Layer-adaptive	$\sim 62.5\%$	91.60	89.82	85.18	48.34	**99.5%**
	POP	$\sim 50\%$	79.96	74.70	63.52	42.72	83.0%
	Uniform	50%	90.57	88.22	83.76	46.06	97.2%
	Layer-adaptive	$\sim 50\%$	91.65	89.36	84.09	47.88	**98.9%**
Llama-3.2-1B	Vanilla	100%	69.07	60.36	58.60	32.39	100.0%
	POP	$\sim 62.5\%$	51.58	40.17	30.87	28.61	70.6%
	Uniform	62.5%	61.95	48.17	41.16	30.17	83.2%
	Layer-adaptive	$\sim 62.5\%$	68.14	63.27	60.96	32.27	**101.8%**
Llama-3.2-3B	Vanilla	100%	83.00	78.53	73.54	43.60	100.0%
	POP	$\sim 62.5\%$	79.26	73.97	66.19	43.28	94.7%
	Uniform	62.5%	72.51	65.59	59.24	38.81	85.1%
	Layer-adaptive	$\sim 62.5\%$	83.95	81.13	77.27	44.97	**103.2%**
	POP	$\sim 50\%$	55.60	44.66	29.88	30.22	58.5%
	Uniform	50%	64.58	56.48	50.06	37.27	75.8%
	Layer-adaptive	$\sim 50\%$	85.05	80.71	76.87	44.53	**103.0%**

As shown in Table 2, the tolerable extent of prefill FFN reduction is model-dependent, with larger models within each family generally showing stronger performance retention at the same reduced prefill widths. For example, on Qwen3-4B, all three allocation strategies remain close to Vanilla at 62.5% retained width, whereas their performance separates more clearly at the tighter 50% budget. This contrast shows that the effect of FFN width allocation becomes more pronounced as the available prefill-width budget decreases. Layer-adaptive allocation remains particularly stable in these settings, while the three strategies exhibit different performance and execution trade-offs discussed in Sec. 3.6.

3.5 General Capability Evaluation

We additionally evaluate whether reduced-width prefill affects general capabilities outside long-context evaluation. Table 3 reports results on MMLU, GSM8K, ARC-Challenge, and HellaSwag across the same model scales and prefill-width settings considered above.

Across all evaluated models and budgets, Uniform Width Reduction and Layer-adaptive Width Reduction retain between 98.0% and 100.6% of the corresponding Vanilla performance on average. This remains the case under the tighter 50% prefill-width setting on Qwen3-4B and Llama-3.2-3B. These results show that substantial reductions in prefill FFN width can largely preserve standard downstream capabilities, even when more noticeable differences emerge on long-context evaluations.

Table 3: General capability under reduced-width prefill. Decode always uses the full FFN width. **Rel.** reports the average performance retention relative to the corresponding Vanilla model across MMLU, GSM8K, ARC-C, and HellaSwag. The highest **Rel.** among reduced-width methods within each matched model–budget setting is shown in **bold**.

Model	Method	Avg. Prefill Width	MMLU	GSM8K	ARC-C	HellaSwag	Rel.
Qwen3-0.6B	Vanilla	100%	43.10	39.88	35.15	44.71	100.0%
	POP	$\sim 62.5\%$	36.33	35.33	33.19	42.74	90.7%
	Uniform	62.5%	43.13	39.65	35.49	44.61	**100.1%**
	Layer-adaptive	$\sim 62.5\%$	44.11	38.21	35.49	44.64	99.7%
Qwen3-4B	Vanilla	100%	68.42	79.23	54.86	64.59	100.0%
	POP	$\sim 62.5\%$	67.19	80.89	54.35	64.79	99.9%
	Uniform	62.5%	67.87	80.82	55.20	65.02	**100.6%**
	Layer-adaptive	$\sim 62.5\%$	68.42	81.27	53.58	64.61	100.1%
	POP	$\sim 50\%$	44.67	75.36	48.98	62.68	86.7%
	Uniform	50%	67.62	80.97	54.95	64.39	**100.2%**
	Layer-adaptive	$\sim 50\%$	67.91	80.36	53.33	64.05	99.3%
Llama-3.2-1B	Vanilla	100%	45.73	36.01	40.27	60.95	100.0%
	POP	$\sim 62.5\%$	43.69	27.98	40.02	59.91	92.7%
	Uniform	62.5%	44.97	35.25	39.59	60.02	98.3%
	Layer-adaptive	$\sim 62.5\%$	45.57	35.56	41.04	60.22	**99.8%**
Llama-3.2-3B	Vanilla	100%	61.02	69.98	52.13	72.44	100.0%
	POP	$\sim 62.5\%$	60.91	67.63	50.17	70.98	97.7%
	Uniform	62.5%	60.56	68.76	51.96	70.61	98.7%
	Layer-adaptive	$\sim 62.5\%$	60.90	68.31	52.22	71.55	**99.1%**
	POP	$\sim 50\%$	60.30	57.54	50.34	69.78	93.5%
	Uniform	50%	59.87	68.39	51.45	70.48	98.0%
	Layer-adaptive	$\sim 50\%$	60.80	68.39	51.88	71.65	**99.0%**

3.6 Discussion of Prefill Reduction Strategies

The three reduction strategies (Uniform Width Reduction, Layer-adaptive Width Reduction, and POP) offer different trade-offs between allocation flexibility and execution regularity. POP uses a coarse-grained binary allocation, allowing selected FFNs to be skipped entirely, but provides limited flexibility in distributing capacity across depth and can incur larger quality degradation under tighter budgets in our experiments. Uniform Width Reduction applies the same width to every layer, resulting in regular per-layer computation and consistent GEMM shapes, but cannot adapt to differences in layer-wise capacity requirements and may also degrade as the budget becomes more constrained. Layer-adaptive Width Reduction provides finer control over how capacity is distributed across depth and retains model quality more consistently under tighter budgets, at the cost of less uniform per-layer computation.

4 Related Work

Efficient prefill. Recent work has explored several ways to reduce computation during prefill. FTP (Xu et al., 2025) skips FFN computation for selected prompt tokens, while FastForward (Gautam et al., 2026) predicts which FFN neurons should be activated for a given input. WIDE (Hu et al., 2026) similarly varies FFN width at the token level and across both prefill and decode, allowing different tokens to use different amounts of intermediate capacity. Unlike FastForward and WIDE, which make input- or token-dependent FFN decisions, our active FFN widths are fixed across inputs at inference time and vary only across network depth. At a coarser granularity, POP (He et al.,

2026) prunes later Transformer layers during prefill, while YOCO (Sun et al., 2024) redesigns the architecture so that later computation can reuse a shared KV representation and be avoided for most prompt positions. Our work studies a complementary, input-independent setting at a finer structural granularity: we reduce FFN width within active Transformer layers and examine how much capacity can be reduced during prefill and how the retained capacity should be distributed across layers.

Dynamic and elastic FFN computation. A separate line of work studies models that can execute with different amounts of FFN capacity. MatFormer (Devvrit et al., 2024) introduces nested FFN subnetworks that support multiple operating widths within a shared set of parameters, while WIDE (Hu et al., 2026) dynamically selects FFN width at the token level. These works establish that Transformers can support multiple FFN operating widths. Building on this nested structure, we use shared-parameter FFN subsets to realize reduced-width prefill while retaining the full FFN for decode. Our focus is on how much FFN capacity can be reduced during prefill without compromising model quality.

5 Conclusion

Our study shows that substantial FFN capacity can be removed during prefill without compromising model quality. The learned allocations further reveal fine-grained redundancy within individual Transformer layers: Earlier layers generally retain more capacity, yet still operate below the full FFN width. These findings extend the view of removable prefill computation beyond coarse-grained layer skipping to computation within individual layers. When the computational budget is sufficiently large, different allocation strategies can achieve comparable performance; as the budget becomes tighter, however, the distribution of retained capacity across layers becomes increasingly important. More broadly, Nested FFN provides a concrete mechanism for studying fine-grained computational redundancy during prefill and motivates extending this perspective to other architectural components, such as mixture-of-experts (MoE) layers.

Limitation

Training corpus. Our adaptation experiments use a single long-context corpus, PG19 (Rae et al., 2019), with the same data, data order, number of steps, and hyperparameters across all configurations. This controlled setting supports relative comparisons between architectural configurations, but we do not study how the results vary with the choice of adaptation corpus. Since PG19 primarily contains narrative prose, extending the analysis to more diverse domains, such as code and other structured content, is an important direction for future work.

Model scale. Although we evaluate multiple model sizes and families, our experiments cover models up to 4B parameters. Extending the study to substantially larger models would provide a broader view of how the tolerable extent of prefill FFN reduction and depth-wise allocation patterns evolve with scale.

Architectural scope. We focus on FFN width as a controllable factor for studying removable computation during prefill. Extending this analysis to other components may reveal additional opportunities for fine-grained prefill reduction.

AI Use Statement

Generative AI tools were used to assist with writing and editing, literature discovery, discussion of experimental design, and drafting portions of the manuscript. All technical claims, experimental decisions, results, citations, and final text were reviewed and verified by the authors, who take full responsibility for the content of the paper.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context

understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI. Deepseek-v4.1-flash: Pushing the limits of KV cache compression, 2026. URL https://arxiv.org/abs/2609.19969.

Devvrit, Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham Kakade, Ali Farhadi, and Prateek Jain. Matformer: Nested transformer for elastic inference, 2024. URL https://arxiv.org/abs/2310.07707.

Aayush Gautam, Mukul Gagrani, Junyoung Park, Mingu Lee, Chiris Lott, and Narasimha Reddy. Fast forward: Accelerating llm prefill with predictive ffn sparsity. *arXiv preprint arXiv:2602.00397*, 2026.

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. *arXiv preprint arXiv:2407.21783*, 2024.

Junhui He, Zhihui Fu, Jun Wang, and Qingan Li. Pop: Prefill-only pruning for efficient large model inference. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 5864–5877, 2026.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Haozhe Hu, Hao Wu, Peiran Yin, Chao Han, Yunpu Ma, and Xiaoyu Shen. Wide: Boosting adaptive llm inference via token-level dynamic width pruning. *arXiv preprint arXiv:2607.28418*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. *Advances in Neural Information Processing Systems*, 37:7339–7361, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020. URL https://arxiv.org/abs/1910.03771.

Guo-Hao Xu, Jingzhen Ding, Huping Ding, Zhao Xu, and Kaifu Zhang. Ftp: Efficient prefilling for long-context llm inference via ffn token pruning. 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

A Comparison with Original POP

To enable a cleaner comparison of different FFN capacity allocation strategies, we use a controlled variant of POP in our main experiments. Table 4 summarizes how this variant differs from Original POP (He et al., 2026). We further compare them applied to the same layers to verify that the modification does not weaken the POP baseline. As shown in Table 5, the POP variant used in our main experiments remains comparable to or stronger than Original POP across the evaluated tasks, with particularly clear advantages on long-context evaluations and under tighter budgets. These results indicate that the controlled variant provides a competitive baseline, supporting the validity of our comparisons among different FFN allocation strategies.

Table 4: Differences between Original POP and the POP configuration used in our main experiments.

Configuration	**Original POP**	**POP**
Skipped computation	Entire layer	FFN only
Attention in removed layers	Skipped	Retained
Full-width suffix window ω	1	512
Depth-wise removal pattern	Binary	Binary
Additional training	No	No

Table 5: Performance of Original POP and the POP configuration used in our main experiments. Both configurations are applied to the same layers within each width setting. Best results within each matched setting are shown in **bold**.

Model	**Method**	**Avg. Prefill**	**Long-context Evaluation**				**General Capability**			
		FFN Width	**RULER 8K**	**RULER 16K**	**RULER 32K**	**LongBench**	**MMLU**	**GSM8K**	**ARC-C**	**HellaSwag**
Qwen3-4B	Original POP	$\sim 62.5\%$	85.58	83.91	79.50	39.55	57.23	79.08	**55.03**	**67.10**
	POP	$\sim 62.5\%$	**89.24**	**86.51**	**82.66**	**47.71**	**67.19**	**80.89**	54.35	64.79
	Original POP	$\sim 50\%$	20.41	15.45	12.17	28.49	33.49	69.75	48.81	**66.18**
	POP	$\sim 50\%$	**79.96**	**74.70**	**63.52**	**42.72**	**44.67**	**75.36**	**48.98**	62.68
Llama-3.2-3B	Original POP	$\sim 62.5\%$	66.31	59.41	50.56	34.85	59.97	54.66	50.00	70.24
	POP	$\sim 62.5\%$	**79.26**	**73.97**	**66.19**	**43.28**	**60.91**	**67.63**	**50.17**	**70.98**
	Original POP	$\sim 50\%$	47.03	40.61	**30.59**	22.05	42.86	42.61	47.53	68.55
	POP	$\sim 50\%$	**55.60**	**44.66**	29.88	**30.22**	**60.30**	**57.54**	**50.34**	**69.78**

B Prompt-Length Statistics for General-Capability Evaluation

Reduced-width prefill is applied only to prompt positions preceding the full-width suffix window of size $\omega = 512$. We therefore examine the lengths of the formatted prompts used in our general-

capability evaluation to verify that these benchmarks meaningfully exercise the reduced-width prefill. Table 6 reports prompt-length statistics under the Qwen and Llama tokenizers after applying the corresponding few-shot examples and prompt formatting.

For both tokenizer families, all prompts in ARC-Challenge, GSM8K, and HellaSwag exceed 512 tokens. For MMLU, 63.47% of prompts exceed 512 tokens under the Qwen tokenizer and 65.88% under the Llama tokenizer. Thus, the reduced-width prefill is exercised by all examples in three of the four benchmarks and by the majority of MMLU examples.

Table 6: Prompt-length statistics for the general-capability benchmarks. Lengths are measured after applying the corresponding few-shot examples and model-specific prompt formatting. The final column reports the fraction of examples whose prompt length exceeds the full-width suffix window of $\omega = 512$.

Tokenizer	**Task**	**Shots**	**# Examples**	**Mean**	**Median**	**Min / Max**	> 512
Qwen	MMLU	5	14,042	758.2	586	302 / 3,158	63.47%
	ARC-Challenge	25	1,172	1,186.4	1,180	979 / 1,471	100.00%
	GSM8K	5	1,319	1,066.3	1,060	674 / 1,707	100.00%
	HellaSwag	10	10,042	926.3	928	598 / 1,259	100.00%
Llama	MMLU	5	14,042	764.4	583	320 / 3,162	65.88%
	ARC-Challenge	25	1,172	1,205.4	1,200	1,000 / 1,490	100.00%
	GSM8K	5	1,319	969.2	964	648 / 1,518	100.00%
	HellaSwag	10	10,042	947.6	949	620 / 1,280	100.00%