PREFIX-LM: PREFIX-INVARIANT TRANSFORMERS FOR WIDTH-ADAPTIVE PROGRESSIVE INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

LLMs require a significant amount of resources during autoregressive generation, but not all tokens are equal. Token prediction is a task of varying difficulty, but monolithic architectures allocate the same compute regardless of the difficulty of prediction. Progressive inference, where representations are incrementally computed and refined as needed, can address this challenge. This has traditionally been achieved through early-exiting methods that operate along the depth dimension of the network. Elastic networks that operate along the width dimension can select a different computational budget per input, but the resulting outputs cannot be progressively refined. We identify the missing property as *prefix invariance*: the intermediate representations of smaller submodels are prefixes of larger ones. To this end, we propose PREFIX-LM: an architectural design that achieves prefix invariance by enforcing block-triangular maps, enabling KV cache sharing and full reuse of intermediate computations. We apply PREFIX-LM at the 1B scale, demonstrating practical speedups and establishing prefix invariance as a useful foundation for adaptive computation.

1 INTRODUCTION

Efficient text generation is a key challenge in deploying Large Language Models (LLMs). These models typically operate *autoregressively*, processing the prompt in parallel during *prefill* and storing its representations in a key-value (KV) cache. During *decode*, the latest token passes through the network, attending to cached representations to predict the next token and extending the cache. Although caching avoids recomputing past representations, generation remains costly because each decoding step requires a full model pass and depends on the token produced by the previous step.

Efforts to reduce this cost build on the observation that many predictions can be handled by smaller, less expensive models. *Routing* exploits this variability by selecting a model for an entire request (Ong et al., 2025) or individual tokens (Zheng et al., 2024), while *cascading* starts with a smaller model and invokes a larger one when its estimated prediction quality is not satisfactory (Narasimhan et al., 2025). *Speculative decoding* uses a smaller model to draft tokens that a larger model verifies in parallel (Leviathan et al., 2023). However, efficiency depends not only on which model evaluates each token, but also on whether computation can be reused across models. Without such reuse, invoking a larger model may require rebuilding cached representations of the accumulated context, even when only the next prediction requires additional capacity.

This motivates *progressive inference*, in which representations are incrementally refined while retaining prior computation (Zhang et al., 2018), enabling reuse both within a prediction and across the context. *Early exiting* provides an established route to progressive inference, allowing predictions at intermediate layers while retaining their computation if deeper execution is needed (Schuster et al., 2022; Elhoushi et al., 2024). While these approaches adapt computation along depth, deeper, narrower Transformers can offer favorable performance–cost trade-offs (Tay et al., 2022). This motivates a complementary question: *Can progressive inference operate along width while retaining the network's full depth?*

Elastic models already adapt computation through nested channel widths (Yu et al., 2019), feed-forward widths (Devvrit et al., 2024), or low-rank components (Zaccone et al., 2026). However, parameter nesting does not ensure representation nesting. In slimmable networks, additional channels alter existing activations and thus subsequent layer inputs (Yu & Huang, 2019). MatFormer shares cached context between a narrower drafter and a wider verifier, but verification still recomputes the

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.

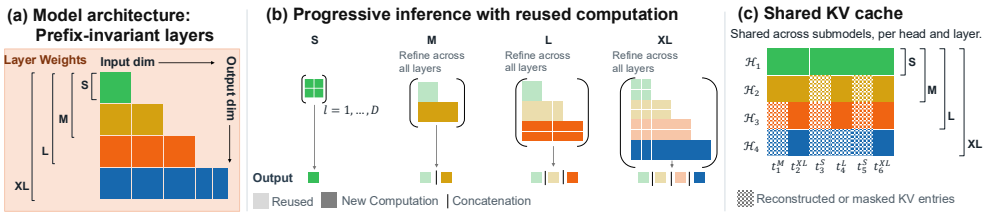


Figure 1: **Progressive inference with PREFIX-LM. (a)** Block lower-triangular weights make every layer prefix invariant: **(b)** escalating from a smaller submodel to a larger one augments existing representations by activating only the next refinement block. **(c)** Shared KV caches can be progressively augmented: lacking KV entries for tokens processed by smaller submodel (superscripts) can be exactly computed (checkerboard), or approximated by the other cache policies of Sec. 3.3.

draft tokens' representations and replaces their cached states (Devvrit et al., 2024). Progressive inference requires preserving existing representations while computing only the missing components.

We identify this property as *prefix invariance*: smaller submodels' representations are exact prefixes of larger ones at every layer. Width expansion thus refines a coherent representation hierarchy, preserving activations and KV-cache blocks while computing missing components. Unlike Matryoshka embeddings, which make final-vector prefixes independently useful (Kusupati et al., 2022), prefix invariance requires agreement across separately executed submodels at every layer.

PREFIX-LM enforces prefix invariance by construction through block-triangular maps, making the first d output coordinates depend only on the first d input coordinates at every layer. This structure, which we prove necessary and sufficient for prefix invariance, allows width expansion to add computation without revising existing representations (see Fig. 1c). In contrast, standard slimmable networks (Yu et al., 2019) fail to attain prefix invariance even when trained with explicit consistency losses, as wider activations can perturb narrower representations (see Fig. 2).

Concretely, we make the following contributions:

- We identify *prefix invariance* as the property that enables progressive width expansion through the exact reuse of activations and KV-cache blocks.
- We introduce PREFIX-LM, an architectural design that enforces prefix invariance via block-triangular maps, and connect it to the truncated SVD (Rippel et al., 2014).
- We empirically show that explicit consistency loss fails to enforce prefix invariance in standard slimmable networks (Yu et al., 2019), whereas PREFIX-LM achieves inference speedups and favorable cost-quality trade-offs through the reuse of hidden states and KV-cache blocks.

2 PRELIMINARIES

This section introduces reusable representations, showing why simply slimming a layer (Yu et al., 2019) does not translate nested parameters to shared representations. The required constraint is prefix invariance, enforced in neural networks by triangular dependencies in the functional map between input and output: the property is preserved under composition, so a model made of such weights is end-to-end prefix-invariant. This property is crucial for progressive inference: we show a realistic experiment where Slimmable Networks fail at achieving it, even augmenting the objective function with prefix-matching losses, while it holds for PREFIX-LM by construction.

From shared parameters to reusable representations. Consider an affine layer $y = Wx + b$, with input and output features partitioned into K ordered blocks. A smaller submodel uses the first s blocks. In the full layer, the corresponding outputs receive contributions from both the retained input blocks and those available only at larger widths:

$$y_{\leq s} = W_{\leq s,\leq s} x_{\leq s} + W_{\leq s,>s} x_{>s} + b_{\leq s}. \tag{1}$$

The cross-block term generally changes existing outputs when additional inputs become active. For a fixed ordered partition, let P_s extract the first s blocks and let $M_s = P_s^\top P_s$ retain those blocks while setting the remaining coordinates to zero.

Definition 2.1 (Prefix invariance)**.** A map $F : \mathbb{R}^D \to \mathbb{R}^E$ is *prefix invariant* with respect to its input and output partitions if, for every input x and budget $s \in \{1, \ldots, K\}$,

$$P_s^{\text{out}} F(x) = P_s^{\text{out}} F(M_s^{\text{in}} x). \tag{2}$$

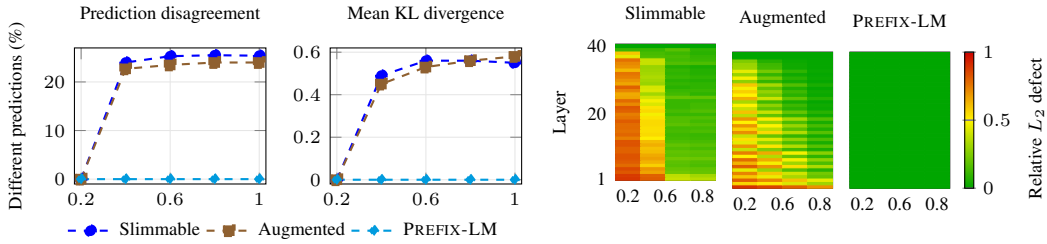


Figure 2: **Slimmable networks are not progressive.** Prediction disagreement (left) and layerwise prefix discrepancy (right) at relative width p for slimmable, an augmented recipe, and PREFIX-LM, on CIFAR-100 and ResNet-20. Despite augmenting the objective with prefix-constraining losses, slimmable networks do not align intermediate representations, accumulating errors over layers

The corresponding width-s operation is the restriction $F^{(s)}(u) = P_s^{\text{out}} F((P_s^{\text{in}})^\top u)$. Prefix invariance ensures that evaluating this restricted operation produces exactly the matching prefix of the full operation: it imposes a simple restriction on how information can flow between blocks.

Proposition 2.2 (Prefix invariance requires triangular maps)**.** *A map $F = (F_1, \dots, F_K)$ is prefix invariant if and only if each output block can be written as $F_r(x) = f_r(x_1, \dots, x_r),\ r = 1, \dots, K$.*

The triangular constraint concerns the complete layer function, relative to the chosen ordering and supported width boundaries. Section 3.1 describes how each operation in a transformer block is modified in PREFIX-LM to make the overall function prefix-invariant.

Composition. Prefix invariance is preserved under composition with aligned intermediate partitions, aligned residual addition, and coordinate-wise nonlinearities. Consider a network of these operations in which each submodel evaluates its corresponding part, starting from a common input embedding. Let $h^{\ell,(s)}(u)$ denote its hidden representation at depth ℓ, evaluated at budget s on input u. Applying prefix invariance successively through the network gives $h^{\ell,(s)}(u) = h^{\ell,(t)}_{\le s}(u),\ s \le t$, for all layers ℓ, i.e., all submodel partitions can be extracted.

Conventional slimmable networks. Standard slimmable networks (Yu et al., 2019) are not prefix-invariant because they retain the cross-block dependencies illustrated above. While their training procedures target useful predictions at multiple widths, they do not impose the prefix identity in Def. 2.1. To further validate that prefix-invariance cannot be imposed onto slimmable networks by properly adjusting the training objective, we propose a realistic experiment in which the standard cross-entropy loss is augmented with prefix-consistency penalties. Concretely, we impose an ℓ_2 penalty between the shared-prefixes of output features of each submodel, across all the layers. This is exactly what block triangular weights in PREFIX-LM enforce by construction. Results in Fig. 2 show that both the standard recipe and the augmented objective function lead to non-negligible prediction disagreement between a progressive-style forward and the actual full recomputation without reuse, corroborating the theoretical result in Proposition 2.2.

3 PREFIX-LM

Here we describe the components of PREFIX-LM, depicted in Fig. 1: (i) the operations of a prefix-invariant transformer block, (ii) the optimized training procedure, matching the FLOPs of a regular one, (iii) efficient KV-cache reuse and approximate alternatives to backfilling for per-token progressive inference. Finally, we discuss the relation to truncated SVD. Proofs are deferred to Sec. C.

3.1 PREFIX-INVARIANT TRANSFORMER BLOCKS

We use K ordered refinement blocks throughout the hidden layers. Block sizes may differ between the residual stream and the feed-forward intermediate representation, but their ordering is aligned: Model s retains blocks $1{:}s$ at every layer. The input embedding remains dense because every budget receives the same raw input. Hidden transformations preserve prefix invariance, and a shared dense classifier accumulates the contributions of the final representation blocks.

Linear maps and feed-forward layers. Following Proposition 2.2, let W_{rq} map input block q to output block r. For a hidden affine map $F(x) = Wx + b$, it is sufficient to set $W_{rq} = 0$ for $q > r$, where W_{rq} maps input block q to output block r or, equivalently, $y_r = \sum_{q=1}^{r} W_{rq} x_q + b_r$. We apply

this ordering to both feed-forward projections and, for gated feed-forward layers, to both the feature and gate branches. Their element-wise product preserves the block structure.

Hierarchical normalization. Standard normalization operators generally violate prefix invariance, since all the feature dimension blocks contribute to the calculation of the statistics. For this reason, for each block r the operation should be constrained to the cumulative feature dimension $D_r = \sum_{q \le r} d_q$. Then, for each token, we define the normalization statistics of block r as:

$$\mu_{:r}(x) = \tfrac{1}{D_r} \textstyle\sum_{i \le D_r} x_i, \qquad \sigma^2_{:r}(x) = \tfrac{1}{D_r} \textstyle\sum_{i \le D_r} (x_i - \mu_{:r}(x))^2. \tag{3}$$

Nested multi-head attention. Multi-Head Attention is modified by partitioning heads into groups $\mathcal{H}_1, \dots, \mathcal{H}_K$; model s activates $\cup_{q \le s} \mathcal{H}_q$, with head dimensions fixed across submodels. Let X contain the normalized embeddings as rows, and write $X_{\le q} = X(P_q^{\text{in}})^\top$. For a head $h \in \mathcal{H}_q$, its query, key, and value projections read only $X_{\le q}$. With a common causal additive mask C, its output and the subsequent block projection are

$$O_h(X) = \operatorname{softmax}\Big(\tfrac{Q_h(X_{\le q}) K_h(X_{\le q})^\top}{\sqrt{d_h}} + C\Big) V_h(X_{\le q}),\ a_r(X) = \textstyle\sum_{q \le r} \operatorname{concat}_{h \in \mathcal{H}_q}[O_h(X)](W^O_{rq})^\top,$$

Each earlier head retains its queries, keys, values, attention weights, and output as later groups are activated. The triangular output projection prevents later head groups from modifying earlier residual-stream blocks. For grouped- or multi-query attention (Ainslie et al., 2023; Shazeer, 2019), the query heads sharing a key/value head must belong to the same group, so that the shared head is computed by the smallest budget that uses it and remains valid for all larger ones.

Cumulative logits. Prefix invariance of features at the last transformer block allows a key property, making progressive inference efficient also for the final vocabulary projection: the output logits can be computed incrementally for each submodel, as activating block $s+1$ adds a logit refinement without recomputing the earlier contributions. More formally, let $h^L = (h^L_1, \dots, h^L_K)$ denote the final representation and partition the shared (dense) classifier as $A = [A_1 \ \cdots \ A_K]$. Its logits are

$$z_s(x) = b + \textstyle\sum_{r=1}^{s} A_r h^L_r(x), \qquad z_{s+1}(x) = z_s(x) + A_{s+1} h^L_{s+1}(x). \tag{4}$$

This is relevant because it avoids repeating the cost of the final head multiplication when escalating to a bigger submodel, a typical overhead of non-exited heads in early-exiting.

Prefix-invariant Mamba blocks For the Nemotron-style Mamba-2 mixer, partition expanded channels into groups $\mathcal{H}_1, \dots, \mathcal{H}_K$ of complete recurrent heads. A head $h \in \mathcal{H}_r$ reads only the residual-stream prefix through block r for its input, gate, and step size. Its full-dimensional B_t, C_t vectors are shared across heads and depend only on the smallest prefix. With causal depthwise-convolved inputs $\bar{x}_{h,t}, \bar{B}_t, \bar{C}_t$ and step size $\Delta_{h,t}$, each head updates its state and output as

$$S_{h,t} = \exp(\Delta_{h,t} A_h) S_{h,t-1} + \Delta_{h,t}\, \bar{x}_{h,t} \bar{B}_t^\top, \qquad o_{h,t} \ = S_{h,t} \bar{C}_t + D_h \bar{x}_{h,t}. \tag{5}$$

The gate, hierarchical RMS normalization, and output projection preserve the same prefix ordering. Existing heads' convolution and recurrent states are therefore width-invariant for a common token history and mask. Escalation still requires backfilling the missing features and states of newly activated blocks through the earlier layers. Full formulas and the argument are in Sec. B.4.

Architectural guarantees. The above constructions lead to prefix-invariant transformers.

Theorem 3.1 (Component invariance)**.** *The affine maps, hierarchical normalization, and nested attention are all prefix invariant.*

Theorem 3.2 (Composability)**.** *Prefix-invariant maps compose when their intermediate partitions are aligned. The class is also closed under aligned addition, coordinate-wise nonlinearities, and concatenation within corresponding blocks.*

Corollary 3.3 (Single-pass exactness)**.** *At every layer of the construction above, the first s blocks of a full-width backbone produce outputs equal to those produced by evaluating submodel s alone, on the same input and with the same attention masks. The cumulative classifier in Eq. (4) consequently recovers every submodel's logits from that pass.*

3.2 Training Prefix-LM

Prefix invariance also shapes how Prefix-LM is trained. Because every submodel's representation is a prefix of the full model's, a single forward pass exposes all K predictions, and any per-budget

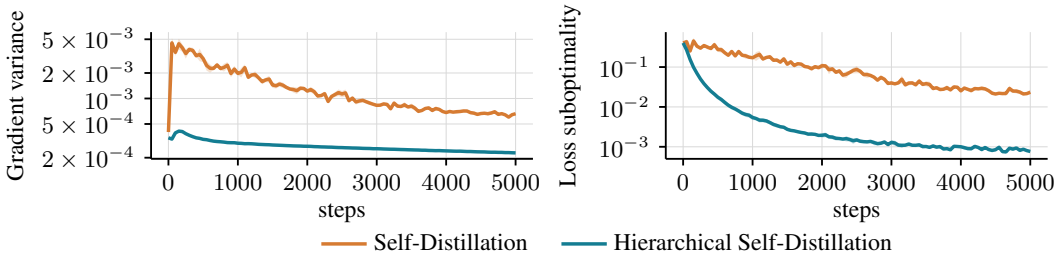


Figure 3: **A closer teacher gives lower-variance gradients.** In nested low-rank regression, distilling each submodel from the next larger one (HSD) rather than from the full model (SD) reduces the variance of the stochastic gradient (left) and, as a result, yields lower suboptimality relative to the per-width optima (right). See Sec. D.2 for details.

objective can be optimized over one shared computation graph. This raises two questions: (i) how to supervise the smaller budgets, and (ii) how to weight the parameters that participate in only a subset of them. We address the first with a *hierarchical self-distillation objective* that trains each submodel from its immediate successor, and the second with a *membership-normalized gradient* that equalizes the participation scaling introduced by joint training. Both use quantities already available from the single pass, so their overhead is confined to the readout.

Training all models in a single pass. By Corollary 3.3, one full-width forward pass yields the logits $z_1, \dots, z_K$ of every submodel, so all per-budget losses can be applied to a single computation graph and backpropagated together. This contrasts with the training of slimmable networks (Yu & Huang, 2019), which introduces submodel sampling at each step. The classifier adds no overhead either, as the cumulative readout in Eq. (4) evaluates each product $A_r h_r^L$ once, so its total cost equals that of a single full-width classifier. The only additional work relative to a vanilla transformer lies in the readout: accumulating prefix logits, normalizing each z_s over the vocabulary, and evaluating K losses instead of one.

Hierarchical Self-Distillation (HSD). A common way to supervise the smaller budgets of a many-in-one model is self-distillation (Yu & Huang, 2019; Zaccone et al., 2026). Beyond transferring predictive knowledge, distillation also serves as a form of partial variance reduction: Safaryan et al. (2023) show that the variance of the student's stochastic gradient scales with the discrepancy between student and teacher predictions. This suggests supervising each submodel from the closest available teacher rather than the full model. In PREFIX-LM, the natural choice is the next larger budget, which differs from the student by exactly one refinement block and therefore provides a target that is both stronger than the student and closer to it than the full model. We call this *hierarchical self-distillation* (HSD). For $K \geq 2$, let $p_s = \mathrm{softmax}(z_s)$ and let sg stop teacher gradients:

$$\mathcal{L}_{\mathrm{HSD}} = \tfrac{1}{K-1} \textstyle\sum_{s=1}^{K-1} D_{\mathrm{KL}}(\mathrm{sg}[p_{s+1}] \,\|\, p_s)\,, \qquad \mathcal{L} = \tfrac{1}{K}\left(\mathrm{CE}(z_K, y) + (K-1)\mathcal{L}_{\mathrm{HSD}}\right). \tag{6}$$

Only the full model receives label supervision; all other budgets follow the detached distribution of their successor, and all teachers come from the same forward pass, so HSD requires no additional teacher model or pass. HSD is also cheap to backpropagate (see Sec. B.1). We visualize its improved gradient variance and convergence in Figure 3 on nested low-rank regression, which is equivalent to solving a truncated SVD along each width (Sec. D.2).

Membership-Normalized Gradients (MNG). Joint training introduces a structural imbalance, as parameters first activated at budget r receive gradients only from the losses of budgets $s \geq r$. Writing ℓ_s for budget s in Eq. (6), the overall gradient with respect to such parameters θ_r is

$$\nabla_{\theta_r} \mathcal{L} = \tfrac{1}{K} \textstyle\sum_{s=r}^{K} \nabla_{\theta_r} \ell_s = c_r \cdot \tfrac{1}{K-r+1} \textstyle\sum_{s=r}^{K} \nabla_{\theta_r} \ell_s, \qquad c_r = \tfrac{K-r+1}{K}, \tag{7}$$

that is, c_r times the mean gradient over the objectives in which θ_r participates. Late blocks are therefore updated with systematically smaller steps, down to a factor $1/K$ for the last one, for no reason other than submodel selection. MNG removes this factor by rescaling each block's gradient by $1/c_r$, thereby acting as a blockwise preconditioner to equalize participation scales.

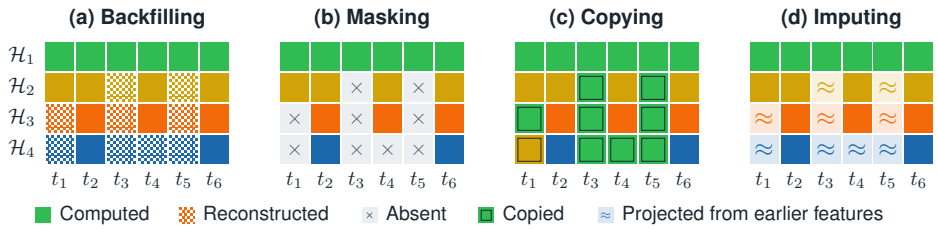


Figure 4: **KV cache strategies during progressive decoding.** Rows denote head groups $\mathcal{H}_r$, and solid cells were computed during the original exits at the time of decoding each token. **(a)** Backfilling reconstructs missing historical KV states. **(b)** Masking performs sparse attention over absent heads. **(c)** Copying reads from KV states that belong to earlier blocks (cycling which block) if the entry is otherwise missing. **(d)** Imputing obtains KV states by applying each missing head's KV projections to the available features at the original exit.

3.3 Autoregressive KV-cache reuse

KV cache reuse in autoregressive language modeling follows as a straightforward consequence of prefix invariance. Switching budgets during decoding, therefore, never invalidates existing cache entries. Methods without prefix invariance offer no such guarantee: their shared heads produce different keys and values at different widths, so a width change forces the receiving model to rebuild or approximate the context (Devvrit et al., 2024; Liu et al., 2026).

However, in routing or model cascades, given the current token processed with budget $t > s$ it is possible that a previous token was processed at budget s, missing cache entries for the heads in $\mathcal{H}_{s+1}, \ldots, \mathcal{H}_t$ (Fig. 1c). Prefix-LM supports a family of policies for these entries (Fig. 4). *Backfilling* is exact: since blocks $1{:}s$ of the past tokens remain valid, only blocks $(s{+}1){:}t$ are evaluated over the context, in parallel, rather than re-prefilling it. The other three avoid this pass and instead alter the attention pattern: *masking* lets each newly activated head attend only to tokens for which it has entries, *copying* substitutes the entries of an existing head, and *imputing* estimates the missing entries from the token's available prefix through the head's own projections (Sec. B.2).

Table 1 compares them on commonsense reasoning under cascading: masking remains close to exact-history cascading while requiring no additional computation, although a modest gap to full-width performance remains on ARC-Easy and HellaSwag. A token processed at a small budget is thus already an adequate summary of itself for the larger heads that later attend to it. This is the premise of KV-cache compression, obtained here without a separate compression step: the smaller widths act as the compressor, and their entries are exactly those the full model would have shared, so nothing has to be reconciled afterward. More details are in Section B.2.

Table 1: **Performance comparison across KV cache strategies.** Commonsense reasoning results for the strategies in Figure 4 at a router threshold of 0.2, compared with full-width Prefix-LM.

Task	Masking	Copying	Imputing	Exact-history cascade	Full-width Prefix-LM
ARC-Easy	49.28	49.16	49.20	49.71	**50.80**
HellaSwag	38.00	37.85	37.93	38.30	**39.75**
PIQA	65.78	65.29	**65.94**	64.91	65.40

3.4 Progressive inference

Algorithm 1 Cascading: Progressive inference for one token

Require: token input, KV cache, stopping rule Accept, budgets $1{:}K$
1: $s \leftarrow 1$; compute block 1 through all layers; $z_1 \leftarrow b + A_1 h_1^L$
2: **while** $s < K$ **and** $\neg$ Accept(z_s) **do**
3: compute block $s{+}1$ through all layers, reusing blocks $1{:}s$ and the cache ▷ no recomputation
4: $z_{s+1} \leftarrow z_s + A_{s+1} h_{s+1}^L$; $s \leftarrow s + 1$ ▷ cumulative logits, Eq. (4)
5: **end while**
6: write blocks $1{:}s$ of this token to the cache
7: **return** $\mathrm{softmax}(z_s)$

A token starts at the smallest budget and grows one block at a time until ACCEPT takes its prediction (Algorithm 1); nothing is recomputed, so the cost is that of the final budget alone (Sec. B.3). The ideal rule is the oracle $\tau^\star(x) = \min\{s : \hat{y}_s(x) = \hat{y}_K(x)\}$, the smallest budget already agreeing with the full model, but it needs $\hat{y}_K$. We instead threshold a confidence score of $p_s = \mathrm{softmax}(z_s)$ (top-1 probability, negative entropy, or top-2 margin in probability or logit space), with per-budget thresholds calibrated on held-out data to a target agreement rate, or train logistic regression (LR) or a two-layer MLP on the scores and the final-layer prefix $h^L_{1:s}$ to predict $\hat{y}_s = \hat{y}_K$, a label the single pass provides at every budget. We observed that LR performed best among these three options.

3.5 IS PREFIX-LM EXPRESSIVE ENOUGH?

The blocks of Sec. 3.1 zero out a large part of every hidden weight, and a natural worry is whether much is lost. Truncated SVD provides a test case: it is the prototypical progressive representation, since increasing the rank appends a component and leaves the others untouched, and it is optimal at every rank. A prefix-invariant network should therefore be able to recover it, and it does. The rank-r truncation of $A^\star = U\Sigma V^\top$ is exactly a width-r prefix of one factorization, a dense map $\Sigma V^\top$ into the singular basis followed by a dense readout U whose columns accumulate as in Eq. (4). This is PREFIX-LM with one hidden layer, and adding depth changes nothing:

Theorem 3.4 (Depth preserves the SVD chain)**.** *For any depth and any set of widths, there is a prefix-invariant linear network with dense endpoints whose full width computes $A^\star$ and whose width-r prefix computes the optimal rank-r truncation, simultaneously for every deployed r.*

The block-triangular constraint thus incurs no cost in the linear case: the class contains the optimal predictor for all widths and depths simultaneously. Beyond the linear case, no such simultaneous optimality can be expected, since the per-width objectives are no longer orthogonal as they are for truncated SVD. What the construction does guarantee is the structural counterpart: later blocks never alter what earlier ones compute, so training a larger submodel cannot degrade a smaller one through the forward pass, which is the interference that many-in-one models must otherwise counteract with sampling schedules and distillation (Yu & Huang, 2019; Devvrit et al., 2024; Zaccone et al., 2026).

4 EXPERIMENTS

Our theoretical results show architectural guarantees of PREFIX-LM. We empirically validate three claims related to the benefits of these properties:

- PREFIX-LM performs competitively on language modeling tasks,
- Cascading with progressive inference enables PREFIX-LM to retain most of its performance while using a fraction of the FLOPs,
- PREFIX-LM is able to achieve practical generation speedups during large-batch decoding.

By validating these claims, we establish the practical benefit of PREFIX-LM and prefix invariance.

Training. We train Llama3.2-1B and Nemotron-Flash 1B using a Token-per-Parameter (TPP) ratio 30, on a data mixture adequately adapted from allal et al. (2025) to fit our shorter, chinchilla-sized training. We train all methods on the same number of tokens, following the respective training recipes.

Evaluation. We evaluate the zero-shot accuracy of models on commonsense datasets commonly used in the literature using the lm-evaluation- harness tool (Gao et al., 2024). For cascading, we train logistic-regression stopping policies on FineWeb-Edu documents, tune per-prefix thresholds on separate documents, and evaluate next-token agreement with the full model against estimated linear-layer FLOPs.

4.1 CASCADING POLICY

We evaluate cascading with the learned router of Sec. 3.4 on next-token prediction on the held-out split of FINEWEB-EDU, measuring per-token agreement with the full model's greedy prediction against per-token FLOPs. Figure 6 reports PREFIX-LM against the fixed widths and the oracle τ^*;

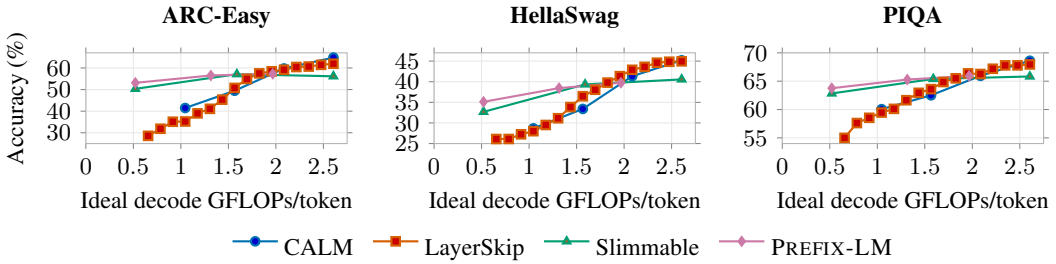


Figure 5: **PREFIX-LM is competitive with early-exit methods.** Accuracy versus ideal decode FLOPs per token for PREFIX-LM, Slimmable networks, and two representative early-exit methods. PREFIX-LM achieves accuracy comparable to Slimmable networks while also being prefix invariant. Both outperform the early-exit methods at low FLOPs budgets.

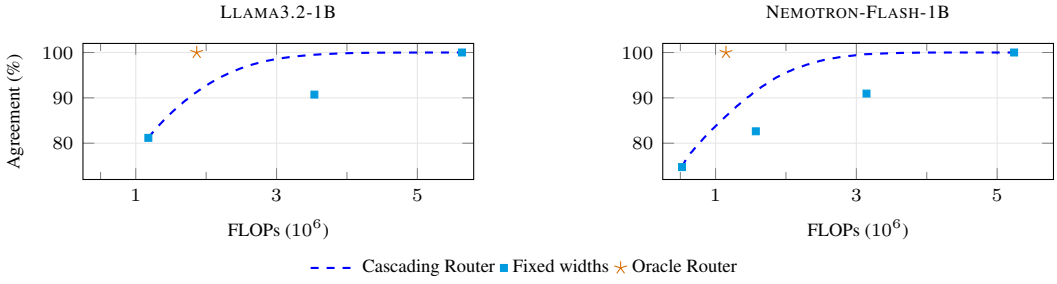


Figure 6: **Agreement versus FLOPs for LLAMA3.2-1B and NEMOTRON-FLASH-1B.** FLOPs are computed for each linear-layer. Agreement is the ratio of samples that match the full model.

the router is fit on 1000 held-out sequences in less than 10 minutes on a single A100 GPU, and its curve is traced by sweeping the acceptance threshold.

Cascading dominates any static choice and closely tracks the full model outputs at much lower costs. This is what prefix invariance is designed to produce: larger budgets refine rather than revise, so the small model's own state reveals when refinement is unnecessary, and the cascade pays only for the budget it stops at.

4.2 CASCADING INFERENCE WALL-CLOCK DECODING SPEEDUP

To verify that we can achieve practical wall-clock speedups with PREFIX-LM when cascading, we benchmark the throughput of large-batch decoding. Large-batch cascading introduces engineering challenges, such as handling heterogeneous cascades within a batch, that make this speedup non-trivial. Using an engineered implementation, we benchmark PREFIX-LM-Llama versus vanilla LLAMA3.2-1B on a single H200. We measure the baseline throughput on vLLM, a strong inference engine (Kwon et al., 2023). We measure batch sizes of 256, 512, and 1024, varying top-2 margin thresholds as described in 3.4, and plot the results in Figure 7. We observe that we are able to realize PREFIX-LM's efficiency gains in scenarios simulating large-batch inference. Experimental details are provided in Section D. Comparison with early-exiting is provided in Fig. 5.

5 RELATED WORK

Elastic models and nested representations. Elastic models make one set of weights usable at several compute budgets. Slimmable networks do so by varying channel width (Yu et al., 2019; Yu & Huang, 2019), and once-for-all networks allow broader architecture choices (Cai et al., 2020). For language models, MatFormer nests feed-forward blocks (Devvrit et al., 2024), while Flextron, LlamaFlex, Star Elastic, and FlexRank offer other forms of shared, variable-capacity models (Cai et al., 2024; 2025; Taghibakhshi et al., 2026; Zaccone et al., 2026). These approaches reduce the cost of maintaining multiple model sizes. However, sharing weights does not imply representation reuse. Increasing the budget can change activations that the smaller model has already produced.

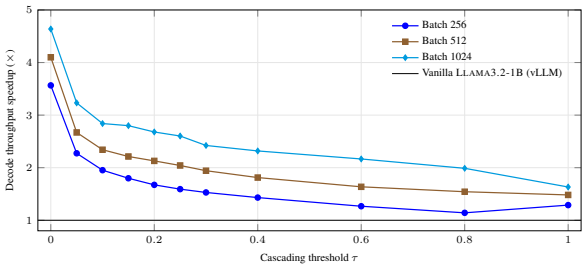


Figure 7: **Decode throughput relative to vanilla LLAMA3.2-1B (vLLM).** We measure decode throughput across various batch sizes in tokens/s on 1,024 FINEWEB-EDU text snippets of 1,024 tokens each, with up to 1,024 generated tokens. We observe speedups with large batch sizes, indicating that PREFIX-LM can achieve practical throughput gains.

PREFIX-LM addresses this distinction by preserving those activations at every layer. This also differs from Matryoshka representations and nested dropout, which make truncated representations useful without requiring separately executed widths to agree throughout the network (Kusupati et al., 2022; Rippel et al., 2014).

Adaptive and progressive inference. Once several budgets are available, inference can choose where to spend them. Routing selects a model for a request or token (Ong et al., 2025; Zheng et al., 2024); cascading tries a cheaper model first and escalates when needed (Narasimhan et al., 2025). Speculative decoding uses a stronger model to verify cheap drafts (Leviathan et al., 2023). The benefit of "escalating" depends on how much of the first model's work survives the switch. Progressive inference seeks precisely this reuse as a prediction is refined (Laskaridis et al., 2020a;b; Zhang et al., 2018). In language generation, CALM and LayerSkip pursue it along depth, while managing the KV states of tokens that exited early (Schuster et al., 2022; Elhoushi et al., 2024; Bae et al., 2023). PREFIX-LM instead adds width while retaining full depth. With a shared token history and attention mask, PREFIX-LM preserves existing KV entries exactly across widths; only missing blocks need to be computed for full-context attention.

Triangular connections. Preserving earlier work as width grows requires controlling which channels can influence each other. Doubly Nested Networks use channel-causal convolutions to support width and depth slicing (Kim et al., 2018), and Any-Width Networks use lower-triangular weights in adjustable-width CNNs (Vu et al., 2020). Unlike prior work, focusing on small CNNs, we formulate the underlying requirement as prefix invariance and enforce it across Transformer operations, including normalization, attention, and residual paths. Our work is the first to extend logits and KV caches incrementally during autoregressive generation, giving true progressive inference for language models.

6 CONCLUSION

We introduced PREFIX-LM, a Transformer whose submodels are prefixes of one another at every layer. This prefix invariance lets additional width refine rather than replace earlier computation, so predictions, hidden states, and KV caches are reused exactly across budgets. The class recovers truncated SVD at any depth in the linear case, and in practice, a learned router cascades billion-parameter models to near full-model agreement at a fraction of the cost, with the savings carrying through to decoding throughput. Closing the remaining gap to the oracle and combining width-wise refinement with depth-wise early exit are natural next steps.

AI USE STATEMENT

We used generative AI tools to assist with implementing and debugging PREFIX-LM's cascading policies and evaluation pipelines, and with optimizing inference code for decoding throughput. These tools also supported analysis of agreement–compute tradeoffs, discussion of theoretical arguments, figure preparation, and manuscript drafting and editing. Implementation checks included automated tests and comparisons with reference outputs. The authors take responsibility for the final methodology, claims, experimental results, and artifacts.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=hmOwOZWzYE.

Loubna Ben allal, Anton Lozhkov, Elie Bakouch, Gabriel Martin Blazquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Agustín Piqueres Lajarín, Hynek Kydlíček, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan Son NGUYEN, Ben Burtenshaw, Clémentine Fourrier, Haojun Zhao, Hugo Larcher, Mathieu Morlon, Cyril Zakka, Colin Raffel, Leandro Von Werra, and Thomas Wolf. SmolLM2: When smol goes big — data-centric training of a fully open small language model. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=3JiCl2A14H.

Sangmin Bae, Jongwoo Ko, Hwanjun Song, and Se-Young Yun. Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5910–5924, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.362. URL https://aclanthology.org/2023.emnlp-main.362/.

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.

Ruisi Cai, Saurav Muralidharan, Greg Heinrich, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. Flextron: Many-in-one flexible large language model. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5298–5311. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/cai24e.html.

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 *International Conference on Learning Representations*, volume 2025, pp. 42250–42263, 2025.

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. In *Advances in Neural Information Processing Systems*, 2024.

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 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, 2024.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Jaehong Kim, Sungeun Hong, Yongseok Choi, and Jiwon Kim. Doubly nested network for resource-efficient inference, 2018. URL https://arxiv.org/abs/1806.07568.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, and Ali Farhadi. Matryoshka representation learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Stefanos Laskaridis, Stylianos I. Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D. Lane. Spinn: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, MobiCom '20, New York, NY, USA, 2020a. Association for Computing Machinery. ISBN 9781450370851. doi: 10.1145/3372224.3419194. URL https://doi.org/10.1145/3372224.3419194.

Stefanos Laskaridis, Stylianos I. Venieris, Hyeji Kim, and Nicholas D. Lane. Hapi: hardware-aware progressive inference. In *Proceedings of the 39th International Conference on Computer-Aided Design*, ICCAD '20, New York, NY, USA, 2020b. Association for Computing Machinery. ISBN 9781450380263. doi: 10.1145/3400302.3415698. URL https://doi.org/10.1145/3400302.3415698.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Yuhan Liu, Yuyang Huang, Jiayi Yao, Shaoting Feng, Zhuohan Gu, Kuntai Du, Hanchen Li, Yihua Cheng, Junchen Jiang, Shan Lu, Madan Musuvathi, and Esha Choukse. *DroidSpeak: KV cache sharing across fine-tuned model variants*. USENIX Association, USA, 2026. ISBN 978-1-939133-54-0.

Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, Seungyeon Kim, Neha Gupta, Aditya Krishna Menon, and Sanjiv Kumar. Faster cascades via speculative decoding. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=vo9t20wsmd.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=8sSqNntaMr.

Oren Rippel, Michael A. Gelbart, and Ryan P. Adams. Learning ordered representations with nested dropout. *arXiv preprint arXiv:1402.0915*, 2014.

Mher Safaryan, Alexandra Peste, and Dan Alistarh. Knowledge distillation performs partial variance reduction. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=tzxP9Rx0LV.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

Noam Shazeer. Fast transformer decoding: One write-head is all you need, 2019. URL https://arxiv.org/abs/1911.02150.

Ali Taghibakhshi, Ruisi Cai, Saurav Muralidharan, Sharath Turuvekere Sreenivas, Aditya Vavre, Ameya Sunil Mahabaleshwarkar, Bilal Kartal, Sheldon Liang, Marcin Chochowski, Zijia Chen, et al. Star elastic: Many-in-one reasoning LLMs with efficient budget control. *arXiv preprint arXiv:2605.07182*, 2026.

Yi Tay, Mostafa Dehghani, Jinfeng Rao, William Fedus, Samira Abnar, Hyung Won Chung, Sharan Narang, Dani Yogatama, Ashish Vaswani, and Donald Metzler. Scale efficiently: Insights from pretraining and finetuning transformers. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=f2OYVDyfIB.

Thanh Vu, Marc Eder, True Price, and Jan-Michael Frahm. Any-width networks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 3018–3026, 2020. doi: 10.1109/CVPRW50498.2020.00360.

Jiahui Yu and Thomas S. Huang. Universally slimmable networks and improved training techniques. In *IEEE/CVF International Conference on Computer Vision*, pp. 1803–1811, 2019.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1gMCsAqY7.

Riccardo Zaccone, Stefanos Laskaridis, Marco Ciccone, and Samuel Horváth. FlexRank: Nested low-rank knowledge decomposition for adaptive model deployment. In *International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=DK0kvnNelx. Spotlight.

Zhi Zhang, Guanghan Ning, Yigang Cen, Yang Li, Zhiqun Zhao, Hao Sun, and Zhihai He. Progressive neural networks for image classification, 2018. URL https://arxiv.org/abs/1804.09803.

Wenhao Zheng, Yixiao Chen, Weitong Zhang, Souvik Kundu, Yun Li, Zhengzhong Liu, Eric P. Xing, Hongyi Wang, and Huaxiu Yao. CITER: Collaborative inference for efficient large language model decoding with token-level routing. In *Adaptive Foundation Models: Evolving AI for Personalized and Efficient Learning*, 2024. URL https://openreview.net/forum?id=0tSUpTUgfP.

A Appendix

B Additional Discussion

B.1 Efficient Execution of Distillation Loss

We exploit the fact that the output head of Prefix-LM is cumulative because it is shared between the submodels,

$$z_i = \sum_{j \le i} \Delta z_j, \qquad p_i = \mathrm{softmax}(z_i).$$

For equally weighted adjacent-prefix distillation with detached teachers and a final hard-label loss, the gradients reaching classifier block i telescope:

$$\nabla_{\Delta z_i} \mathcal{L} = \frac{1}{K} \left[\sum_{j=i}^{K-1} (p_j - p_{j+1}) + p_K - y \right] = \frac{p_i - y}{K},$$

where y is the one-hot target.

For hard-label HSD, the compiled CUDA path also simplifies the backward without changing the objective. If z_i is the cumulative logits at prefix i and $p_i = \mathrm{softmax}(z_i)$, detached-teacher HSD produces successive logit gradients $p_i - p_{i+1}$, followed by $p_K - y$. A disjoint projection block appears in every cumulative prefix from its own prefix onward, so those gradients telescope to $p_i - y$ for that block. The implementation therefore returns the true HSD scalar while attaching the equivalent gradient from one hard-label cross-entropy per block, with preceding cumulative logits detached. This preserves the scalar and first-order parameter gradients used by the optimizer.

B.2 KV Cache Policies During Decoding

Let s_i be the budget used to process token i, and let KV head h belong to block r. Suppressing layer indices, write $\bar{h}_{i,1:s_i}$ for the available normalized features and R_i for rotary encoding at the original position. Prompt tokens are processed at full width. Each policy specifies stored KV entries and the positions $\mathcal{I}_{h,t}$ over which attention is normalized.

Backfilling (exact reconstruction). Reconstruct missing historical feature blocks through all layers before evaluating a larger budget. Since blocks $1{:}s$ of those tokens are already available and remain valid, this requires evaluating only blocks $s{+}1{:}t$ over the context, rather than re-prefilling it. This policy ensures no approximation is made during ifnerence, conditioned on the sequence.

Masking (sparse history). We perform regular attention over heads whose previous entries are all filled, and sparse attention for heads that are missing some entries, depending on which tokens in the sequence cascaded to which budget:

$$\mathcal{I}_{h,t} = \{i \le t : s_i \ge r\}, \qquad (\tilde{k}_{h,i}, \tilde{v}_{h,i}) = (k_{h,i}, v_{h,i}).$$

Missing positions are excluded from the softmax.

Copying (using smaller KV cache). Let $H(s_i)$ be the number of computed KV heads, indexed from one. For each missing head, cyclically select an available head at the same layer and position:

$$\phi_i(h) = 1 + ((h-1) \bmod H(s_i)), \qquad (\tilde{k}_{h,i}, \tilde{v}_{h,i}) = (k_{\phi_i(h),i}, v_{\phi_i(h),i}).$$

Computed entries remain unchanged; the heads of larger models read from the KV cache of smaller ones. Copied keys already include rotary encodings and do not need to be applied again.

Imputing (partial projection). For each missing head in block $r > s_i$, use its own projections restricted to available features:

$$\tilde{k}_{h,i} = R_i W^K_{h,1:s_i} \bar{h}_{i,1:s_i}, \qquad \tilde{v}_{h,i} = W^V_{h,1:s_i} \bar{h}_{i,1:s_i}.$$

This zero-extends the already-normalized features, and computed entries remain unchanged, similarly to copying.

Table 2: **Idealized per-token costs.** Add F_s to obtain total matrix-product FLOPs; multiply the KV columns by $2bN$ to obtain bytes. Reads count logical accesses, not physical HBM traffic. Copying aliases existing entries rather than storing duplicates.

Policy	FLOPs beyond F_s	KV reads/ $(2bN)$	KV writes/ $(2bN)$
Backfilling	$4NtD_s + \Delta F_t$	$tU_s + \sum_{(i,r)\in\mathcal{B}_t} iv_r$	$U_s + \sum_{(i,r)\in\mathcal{B}_t} v_r$
Masking	$4N\sum_{r\le s} d_r m_{r,t}$	$\sum_{r\le s} v_r m_{r,t}$	U_s
Copying	$4NtD_s$	tU_s	U_s
Imputing	$4NtD_s + 4ND_s(U_K - U_s)$	tU_s	U_K

B.3 Idealized Cost of Progressive Inference

Progressive inference computes each reached block once; restarting at every width repeats block r a total of $s - r + 1$ times for exit s. Let

$$F_s = \sum_{r\le s}(C_r + E_r)$$

denote the current token's non-attention backbone and incremental shared-readout FLOPs.

For N layers, let d_r, v_r be the residual/query and KV widths added by block r, and let b be the bytes per cached scalar. With t including the current token and $s = s_t$, define

$$D_s = \sum_{r\le s} d_r, \qquad U_s = \sum_{r\le s} v_r, \qquad m_{r,t} = \left|\{i \le t : s_i \ge r\}\right|.$$

For backfilling, a_i is token i's largest cached exact budget:

$$\mathcal{B}_t = \{(i,r) : i < t,\ a_i < r \le s\}, \qquad \Delta F_t = \sum_{(i,r)\in\mathcal{B}_t} (C_r + 4Nid_r).$$

Reconstruction assumes retained intermediate activations and requires no historical readouts. Counts use two FLOPs per multiply–add and omit elementwise work, routing, and non-KV traffic; physical traffic and runtime depend on the implementation.

B.4 Mamba Block Formulation and State Reuse

Let the expanded channels be partitioned into groups of complete recurrent heads $\mathcal{H}_1, \dots, \mathcal{H}_K$, with fixed head dimension d_h and state dimension N. Budget s activates $\bigcup_{r\le s}\mathcal{H}_r$. Write u_t for the prefix-invariant, pre-normalized residual stream at token t. For $h \in \mathcal{H}_r$, triangular projections of $u_{t,\le r}$ produce a gate $z_{h,t}$, recurrent input $x_{h,t} \in \mathbb{R}^{d_h}$, and step input $\delta_{h,t}$. The projections for the shared vectors $B_t, C_t \in \mathbb{R}^N$ read only $u_{t,\le 1}$; unlike the head channels, their state dimensions are retained in full at every budget.

Let DWConv be a causal depthwise convolution applied separately to x, B, and C, and let $\phi = \mathrm{SiLU}$. The scan inputs and head-specific coefficients are

$$\begin{aligned} \bar{x}_{h,t} &= \phi(\mathrm{DWConv}_h(x_h)_t), \quad \bar{B}_t = \phi(\mathrm{DWConv}_B(B)_t), \quad \bar{C}_t = \phi(\mathrm{DWConv}_C(C)_t), \\ \Delta_{h,t} &= \mathrm{softplus}(\delta_{h,t} + b_h), \quad A_h = -\exp(a_h), \quad D_h \in \mathbb{R}. \end{aligned} \tag{8}$$

Here a_h, b_h, D_h are head-specific parameters. From an initial state $S_{h,0} \in \mathbb{R}^{d_h\times N}$, each active head follows Eq. (5). The gate acts before output normalization: concatenate the head outputs and gates in head order and set $v_t = o_t \odot \phi(z_t)$. If M_r is the number of expanded channels through group r, hierarchical RMS normalization and the block-triangular output projection give, for channel i in group r,

$$\widehat{v}_{t,i} = \gamma_i \frac{v_{t,i}}{\sqrt{M_r^{-1}\sum_{j\le M_r} v_{t,j}^2 + \epsilon}}, \qquad m_{t,r} = \sum_{q\le r} W^O_{rq}\widehat{v}_{t,q}, \qquad h'_{t,r} = h_{t,r} + m_{t,r}. \tag{9}$$

For any budgets $s \le s'$, consider a head $h \in \mathcal{H}_r$ with $r \le s$. Its projected x, z, δ agree because $u_{t,\le r}$ agrees by prefix invariance; its shared B, C agree because $u_{t,\le 1}$ agrees. Depthwise convolution uses only that head's channels or the shared B, C channels, so its convolution history and barred inputs also agree by induction on tokens. Starting from identical recurrent states, Eq. (5) then gives identical $S_{h,t}$ and $o_{h,t}$ at every token. At masked positions, neither the convolution histories nor the recurrent states advance. Finally, the gate and the fixed-boundary statistics in Eq. (9) leave earlier normalized groups unchanged, while its triangular projection prevents later heads from modifying earlier residual blocks. Thus the complete Mamba residual block is prefix invariant for the same token history and padding mask. On a width increase, existing convolution and recurrent states remain valid. Exact escalation must backfill the missing residual features and newly activated heads' convolution and recurrent states over earlier tokens, proceeding layer by layer because deeper mixers depend on the new outputs of earlier layers.

C Proofs

C.1 Deep linear networks and truncated SVD

Throughout, $A^\star = U\Sigma V^\top$ is a compact SVD with $\Sigma = \mathrm{diag}(\sigma_1, \dots, \sigma_k)$, $\sigma_1 \ge \cdots \ge \sigma_k > 0$, and $A_r = U\Sigma M_r V^\top$ is its rank-r truncation, where M_r is the prefix mask of Sec. 2 with single-coordinate blocks. Equation

$$A_r x = U\, M_r\, (\Sigma V^\top x) = \sum_{i=1}^{r} u_i \big(\sigma_i v_i^\top x\big), \tag{10}$$

shows that a network with dense input map $\Sigma V^\top$, one hidden layer of width k, and dense readout U realizes every A_r as a prefix. The question is whether inserting hidden layers, which must be block-triangular to preserve prefix invariance, restricts this. Thm. 3.4 states that it does not: at any depth, and for any choice of the hidden maps but one, the network still computes $A^\star$ at full width and A_r at every deployed prefix.

For deployed budgets $\mathcal{K} = \{r_1, \dots, r_J\}$ with $r_J = k$, define the block-triangular class

$$\mathcal{L}_\mathcal{K} = \big\{G \in \mathbb{R}^{k\times k} :\ M_r G(I - M_r) = 0 \text{ for all } r \in \mathcal{K}\big\}. \tag{11}$$

Lemma C.1 (Closure). *$\mathcal{L}_\mathcal{K}$ is closed under products and under inversion of its invertible elements, contains every diagonal matrix, and satisfies $M_r G = M_r G M_r$ for all $G \in \mathcal{L}_\mathcal{K}$ and $r \in \mathcal{K}$.*

Proof. The last identity restates equation 11. For products, $M_r GH(I - M_r) = M_r G M_r H(I - M_r) + M_r G(I - M_r)H(I - M_r) = 0$, since $M_r H(I - M_r) = 0$ and $M_r G(I - M_r) = 0$. For inverses, fix $r \in \mathcal{K}$ and write G in block form with respect to $\mathbb{R}^k = \mathbb{R}^r \oplus \mathbb{R}^{k-r}$; then $G = \left[\begin{smallmatrix} G_{11} & 0 \\ G_{21} & G_{22} \end{smallmatrix}\right]$ with G_{11}, G_{22} invertible because $\det G = \det G_{11} \det G_{22}$, and $G^{-1} = \left[\begin{smallmatrix} G_{11}^{-1} & 0 \\ -G_{22}^{-1} G_{21} G_{11}^{-1} & G_{22}^{-1} \end{smallmatrix}\right]$ has the same zero block. Diagonal matrices satisfy equation 11 trivially. □

Proof of Thm. 3.4. We first treat hidden widths $d_1 = \cdots = d_L = k$. For $L = 1$ take $G_1 = \Sigma$, recovering the factorization in equation 10. For $L \ge 2$, choose arbitrary invertible $G_1, \dots, G_{L-1} \in \mathcal{L}_\mathcal{K}$ and set $G_L = \Sigma(G_{L-1} \cdots G_1)^{-1}$, which lies in $\mathcal{L}_\mathcal{K}$ by Lemma C.1. The network is $x \mapsto UG_L \cdots G_1 V^\top x$ with dense endpoints $V^\top$ and U, and its full-width product is $U\Sigma V^\top = A^\star$.

For the prefix claim, the width-r submodel retains the first r coordinates after every hidden map and reads out through the first r columns of U, i.e., computes $UM_r G_L M_r G_{L-1} \cdots M_r G_1 V^\top x$. By Lemma C.1, $M_r G_L M_r = M_r G_L$, so $M_r G_L M_r G_{L-1} = M_r(G_L G_{L-1})$; since products stay in $\mathcal{L}_\mathcal{K}$, repeating the step gives $M_r G_L \cdots G_1 = M_r \Sigma = \Sigma M_r$. Hence the prefix computes $U\Sigma M_r V^\top x = A_r x$ for every $r \in \mathcal{K}$. The same identity shows that feeding G_1 the full $V^\top x$ or its prefix $M_r V^\top x$ gives the same result, so the dense input map need not be masked.

For general widths $d_\ell \ge k$, let $E_\ell \in \mathbb{R}^{d_\ell \times k}$ embed $\mathbb{R}^k$ into the first k coordinates of layer ℓ, and define $\widetilde{W}_1 = E_1 G_1 V^\top$, $\widetilde{W}_\ell = E_\ell G_\ell E_{\ell-1}^\top$ for $2 \le \ell \le L$, and readout $UE_L^\top$. Each $\widetilde{W}_\ell$ places G_ℓ in its top-left $k \times k$ block and is zero elsewhere, hence block-triangular for every boundary $r \le k$. Since $E_\ell^\top E_\ell = I_k$, the unmasked product is $UG_L \cdots G_1 V^\top = A^\star$. Writing $M_r^{(\ell)}$ for the prefix

mask on layer ℓ, we have $M_r^{(\ell)} E_\ell = E_\ell M_r$ and $E_\ell^\top M_r^{(\ell)} = M_r E_\ell^\top$, so the masked product reduces to the width-k case and equals A_r. □

The theorem shows that depth is free in the linear case: the dense endpoints fix the singular basis, and within that basis any invertible block-triangular trunk can be compensated by one further block-triangular factor. The simplest instance takes $G_1 = \cdots = G_{L-1} = I$ and $G_L = \Sigma$, or symmetrically $\Sigma^{1/2}$ on each endpoint with identity hidden maps, so that each hidden coordinate carries one singular direction and widening appends $\sigma_{r+1} u_{r+1} v_{r+1}^\top x$ without altering the rest.

Lemma C.2 (Squared loss with general input covariance)**.** *Let $\mathcal{R}(M) = \mathbb{E}\|y - Mx\|_2^2$ with $C_x = \mathbb{E}[xx^\top] \succ 0$, and let $A^\star$ minimize $\mathcal{R}$. Then for every M, $\mathcal{R}(M) - \mathcal{R}(A^\star) = \|(M - A^\star)C_x^{1/2}\|_F^2$. Consequently, if $A^\star C_x^{1/2} = U\Sigma V^\top$, the minimizer of $\mathcal{R}$ over rank $\le r$ is $A_r C_x^{-1/2}$ with excess risk $\sum_{i>r} \sigma_i^2$.*

Proof. The normal equations give $\mathbb{E}[(y - A^\star x)x^\top] = 0$, so $\mathcal{R}(M) = \mathcal{R}(A^\star) + \mathrm{tr}\big((M - A^\star)C_x(M - A^\star)^\top\big)$, which is the stated identity. Since $C_x^{1/2}$ is invertible, $\mathrm{rank}(MC_x^{1/2}) = \mathrm{rank}(M)$, so minimizing over rank $\le r$ is the best rank-r approximation of $A^\star C_x^{1/2}$, given by Eckart–Young. □

Applying Thm. 3.4 to $A^\star C_x^{1/2}$ and composing the input map with $C_x^{-1/2}$, which remains dense, yields a prefix-invariant deep linear network whose width-r prefix attains the optimal excess risk at every deployed r; this is the risk statement following the theorem in the main text.

D Experimental Setting

D.1 Training Details

For Llama3.2-1B and Nemotron-Flash-1B we train on a data mixture inspired by the one proposed by allal et al. (2025), scaled down and adapted according author recommendations from 11T to 30B tokens, comprising 60% FineWeb-Edu, 20% Stack-Edu, 15% FineMath-4+, and 5% Cosmopedia-v2. For Prefix-LM, unless otherwise specified, we train using Hierarchical Self-Distillation (HSD) and Membership-Normalized Gradients (MNG), and choose the same nested widths both for Prefix-LM and Slimmable networks. For Llama3.2-1B we choose the nested widths corresponding to using 3,6 and 8 over the total 8 attention/KV cache heads; for Nemotron-Flash-1B we train nested widths corresponding to all of 4 heads.

Over all the main experiments, we fix a sequence length of 2048 and global batch size of 1024, and use AdamW optimizer with standard parameters and learning η rate separately tuned for each method. We found the best ones being consistent across the methods, being $\eta = 5e-5$ for both architectures. We employ a Warmup-Schedule-Decay (WSD) schedule with a warmup of 500 steps and a linear cooldown phase to 0 in the last 10% of training, following the practice of allal et al. (2025).

D.2 Hierarchical distillation on nested low-rank regression

We study a neural network with one hidden layer of 12 neurons, with trainable linear input and output layers and no activations or biases. Each submodel uses a prefix of the hidden neurons, yielding 12 nested predictors with shared parameters. Inputs are sampled independently and uniformly from the unit sphere in 12 dimensions. Targets are generated by a fixed diagonal linear map whose singular values are equally spaced from 1.5 to 0.5, without added observation noise. Under this isotropic input distribution, the optimal population predictor at each width is the corresponding truncated SVD of the target matrix. These optima can be represented simultaneously by prefixes of a single network.

We compare self-distillation from the current full model (SD) with hierarchical self-distillation from the next larger submodel (HSD). Teacher predictions are detached during gradient computation, whereas the full model is supervised with the true targets in both methods. For this regression

experiment, distillation uses half-squared Euclidean prediction error, and the training objective averages the losses equally across all widths. We optimize both factor matrices using plain SGD with minibatch size 8 and a constant learning rate. Initial parameter entries are sampled independently from a zero-mean Gaussian distribution with standard deviation 0.12.

We select a separate learning rate for each method via a powers-of-two grid search, minimizing the mean final population suboptimality after 5,000 updates across three tuning seeds. The grid is extended whenever the minimizing candidate lies at a boundary. We then evaluate the methods on five additional seeds, with matched initial parameters and input streams across methods. Figure 3 reports the comparison over 5,000 SGD updates.

Population suboptimality is computed exactly as the population half-squared prediction loss above the corresponding truncated-SVD optimum, averaged over all widths. Curves show means over the five evaluation seeds, and shaded regions indicate one standard error.

D.3 Implementing High Throughput Cascading for Prefix-LM

While the theoretical FLOPs speedups are large, achievable practical speedup will differ due to engineering overhead. In particular, for batch sizes greater than 1, special care has to be taken when different batch elements route to different levels of the cascade, and some tokens finish their computation long before the rest. Naïve implementations that ignore this issue may not be able to effectively realize overall speedups on these fragmented batches due to underutilized hardware. In general, reducing FLOPs does not always translate to improvements in throughput, so it is important to measure that speedups are possible.

As a proof-of-concept that Prefix-LM enables practical speedups, we task GPT-6 Astra using Codex with engineering a system that optimizes the cascading structure to realize throughput gains. This process involved providing the AI system with un-optimized reference code as a baseline for correctness, and specifying the goal of optimizing throughput while maintaining correctness with respect to the reference. We iterated this optimization process for tens of iterations, ending up with an efficient implementation that performs the following to achieve the throughput reported in Figure 7:

- **Incremental reuse:** cache per-layer activations and cumulative logits as inference cascades. Cascading to a new stage requires only computing newly introduced channels.
- **Batch compaction:** group together batch elements that cascade to later stages, while keeping KV histories in place. This avoids executing later stages over the entire original batch or moving complete KV histories whenever the active subset changes.
- **Packed sparse attention:** apply custom attention kernels perform attention only over KV states which exist for historical tokens. This avoids loading masked KV states.
- **Kernel fusion:** prepare contiguous stage-specific weight matrices, combining the query, key, and value projections into one matrix multiplication and the SwiGLU gate and up projections into another. Contiguous tensors make memory reads and writes more efficient for GPU hardware.
- **CUDA Graph replay:** reuse captured stage computations across decoding steps with matching active-batch sizes, reducing host-side dispatch overhead.

Numerical precision. Implementation uses FP16 for storage and FP32 for accumulation of logits and attention. The use of FP16 over BF16 for cascaded inference is important because intermediate computations face numerical issues during accumulation of intermediate accumulations; accumulations over a subset of channel widths may round differently than an accumulation over the entire width of the model. We observed that BF16 exacerbates these issues.

D.4 Measuring Throughput

We benchmark our Prefix-LM Llama3.2-1B on one NVIDIA H200 using FP16 weights and KV caches with FP32 logit accumulation. The $K = 3$ budgets have cumulative widths $(D_1, D_2, D_3)/D_K = (0.375, 0.75, 1)$. We use 1,024 distinct FineWeb-Edu prompts of 1,024 to-

Table 3: **Decode speedup over vanilla LLAMA3.2-1B (vLLM).** We measure decode throughput across various batch sizes in tokens/s on 1,024 FINEWEB-EDU text snippets of 1,024 tokens each, with up to 1,024 generated tokens. Generations are performed using the cascading policy described in Section 3.4. We observe speedups at large batch sizes, indicating that PREFIX-LM is able to achieve practical throughput gains. $\pm$ denotes measurement error by the maximum observed deviation from the central speedup across two timing measurements per cascade.

	PREFIX-LM-Llama speedup over vanilla LLAMA3.2-1B (vLLM), $\times$		
Threshold τ	Batch 256	Batch 512	Batch 1024
0.00	3.57 ± 0.09	4.10 ± 0.04	4.64 ± 0.08
0.05	2.27 ± 0.06	2.67 ± 0.02	3.23 ± 0.06
0.10	1.95 ± 0.05	2.34 ± 0.02	2.84 ± 0.10
0.15	1.80 ± 0.02	2.21 ± 0.04	2.80 ± 0.07
0.20	1.67 ± 0.02	2.13 ± 0.04	2.68 ± 0.06
0.25	1.59 ± 0.03	2.04 ± 0.04	2.60 ± 0.07
0.30	1.53 ± 0.02	1.94 ± 0.02	2.42 ± 0.05
0.40	1.43 ± 0.02	1.81 ± 0.02	2.32 ± 0.02
0.60	1.27 ± 0.01	1.64 ± 0.03	2.17 ± 0.02
0.80	1.14 ± 0.01	1.54 ± 0.08	1.99 ± 0.08
1.00	1.29 ± 0.02	1.48 ± 0.05	1.63 ± 0.04

Table 4: **Vanilla Llama-3.2 1B (vLLM) decode throughput.** Mean $\pm$ standard deviation over 16 measurements per batch size.

Batch size	Decode throughput (tokens/s)
256	$23{,}264 \pm 236$
512	$25{,}129 \pm 528$
1024	$25{,}929 \pm 455$

kens each, with generation until EOS or 1,024 output tokens. Batch sizes are $B \in \{256, 512, 1024\}$. Completed requests are not replaced until their cohort finishes.

Cascading accepts the first budget satisfying $\tilde{m}_s(x) \geq \tau$, or budget K otherwise, using the probability margin defined in Sec. 3.4. We sweep $\tau \in \{0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.6, 0.8, 1\}$, with endpoints selecting budgets 1 and K. Prefill uses budget K and decoding uses *masking* as the KV cache strategy, avoiding reconstructing the KV cache for historical tokens which exited early.

Decode throughput is

$$\mathrm{TPS}_{\mathrm{dec}} = \frac{\sum_i (n_i - 1)}{\sum_c t_c^{\mathrm{dec}}},$$

where n_i is request i's emitted-token count and t_c^{dec} is cohort c's CUDA-synchronized wall-clock decode time. Prefill and the first output from the prompt state are excluded in order to measure warmed decode time and simulate a production environment.

The baseline is vanilla LLAMA3.2-1B in vLLM 0.30.0 (Kwon et al., 2023) with CUDA Graphs and FP16, using matched prompts, batch sizes, stopping conditions, and fixed-cohort scheduling. Implementations run separately on the same GPU. Each configuration receives one complete warmup and two measured runs with fresh KV state and caching of responses and prefixes disabled. Speedup is calculated by dividing mean cascading throughput by mean matched vLLM throughput.