SNLP: LAYER-PARALLEL INFERENCE VIA STRUCTURED NEWTON CORRECTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoregressive language models execute Transformer layers sequentially, creating a latency bottleneck that is not removed by conventional tensor or pipeline parallelism. We study whether this layerwise dependency can be relaxed by treating the hidden-state trace across layers as the solution of a nonlinear residual equation and solving it with parallel Newton-style updates. While this view is principled, exact Newton corrections require expensive Jacobian-vector products and naive fixed-point iterations are unstable on trained Transformers. We introduce Structured Newton Layer Parallelism (SNLP), a training and inference framework that replaces exact layer Jacobians with cheap architecture-induced surrogate dynamics. In residual Transformers, this yields Identity Newton (IDN), where the correction reduces to a prefix-sum-like update; in mHC-style architectures, HC Newton (HCN) uses the model's residual mixing matrix. We also study SNLP-aware training, including pretraining regularization and direct SNLP-forward SFT. Experiments on Nanochat-scale Transformers show that SNLP exposes a practical speed-quality frontier: on 0.5B models, it reaches up to $2.58\times$ wall-clock speedup, and a less aggressive configuration reaches $1.40\times$ speedup without increasing PPL. The useful tradeoff comes from the biased finite-iteration computation induced by IDN/HCN rather than exact recovery of the sequential trace. We further show that SNLP-aware training mitigates downstream accuracy loss, and that SNLP can serve as a drafter for self-speculative decoding while a sequential verifier preserves output correctness.

1 INTRODUCTION

Transformer language models (Vaswani et al., 2017) are sequential in two distinct senses. Token generation is autoregressive (Radford et al., 2019; Brown et al., 2020), but even for a fixed token prefix, the hidden state must normally pass through the network one layer at a time. Tensor parallelism (Shoeybi et al., 2019), pipeline parallelism (Huang et al., 2019), kernel fusion (Dao et al., 2022), batching, KV caching (Kwon et al., 2023), and speculative decoding (Leviathan et al., 2023; Chen et al., 2023) improve the efficiency of each layer or token step, but they do not remove the layer dependency chain. As models become deeper (Kaplan et al., 2020; Touvron et al., 2023) and decoding remains latency-sensitive, this depthwise dependency becomes a natural target for algorithmic parallelism.

A principled way to expose such parallelism is to view the entire sequence of hidden states across layers as the solution of a nonlinear residual equation. This is analogous to DEER-style (Lim et al., 2024) parallelization of nonlinear recurrences, where Newton iterations solve for all states in a chain jointly rather than executing the chain strictly left-to-right (Danieli et al., 2023). Applied along the depth axis, this perspective suggests that many Transformer layer states could be updated in parallel. However, exact Newton updates require the Jacobian of each full layer with respect to its input. For language-model hidden states, these Jacobians are too large to materialize, and even Jacobian-vector or finite-difference approximations can consume the latency budget that layer parallelism is meant to save. Cheap fixed-point or Jacobi iterations avoid this cost (Song et al., 2021; Santilli et al., 2023), but are often unstable or slow on trained residual networks.

We introduce **Structured Newton Layer Parallelism** (SNLP), a training and inference framework that makes this Newton view practical by replacing exact layer Jacobians with cheap structured

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.

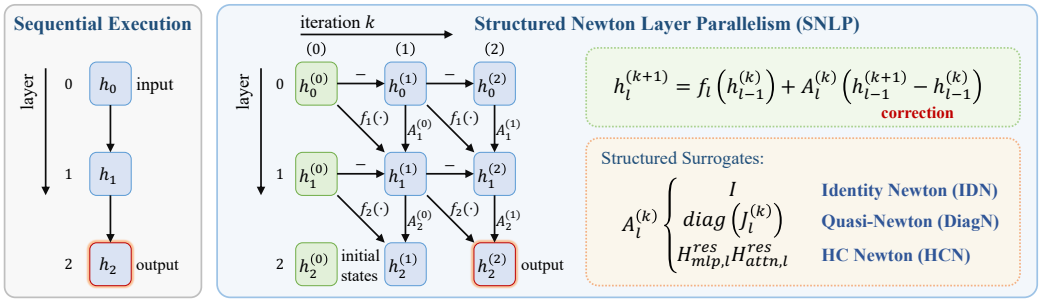

Figure 1: Structured Newton Layer Parallelism (SNLP) replaces sequential layer execution with iterative layer-parallel updates. At each iteration, layer states are updated using the block function f_l and a cheap structured Newton surrogate $A_l^{(k)}$, instantiated as IDN, HCN, or DiagN depending on the architecture.

surrogates. In residual Transformers, the identity residual path gives the simplest surrogate, yielding **Identity Newton** (IDN): the correction reduces to additive prefix-style propagation over depth. For HC/mHC-style models (Zhu et al., 2025; Xie et al., 2025), the architecture exposes a learned residual mixing matrix, yielding **HC Newton** (HCN). **Diagonal Newton** (DiagN) connects SNLP to quasi-DEER and scan-based linear recurrences (Gonzalez et al., 2024), but its Jacobian-estimation cost makes practical speedup more demanding. In all cases, the expensive nonlinear layer or chunk forwards are parallelizable, while the Newton correction is a lightweight structured recurrence.

The second ingredient is training co-design. A pretrained sequential model need not be compatible with a cheap surrogate Jacobian, so we study SNLP-aware training in two forms: a pretraining regularizer that matches finite-iteration SNLP states to the sequential trace, and direct SNLP-forward SFT that trains the approximate computation used at inference. These objectives encourage layer dynamics that are easier to solve with the chosen surrogate and allow task adaptation to preserve SNLP-compatible behavior.

SNLP targets latency-sensitive inference with spare parallel capacity, trading compute and memory for a shorter layerwise critical path. On trained-from-scratch Nanochat-scale models, SNLP exposes a speed-quality frontier: aggressive configurations achieve large wall-clock speedups at the cost of PPL loss, while less aggressive configurations recover near-sequential quality. At the 0.5B scale, SNLP reaches up to $2.58\times$ speedup, and one IDN-regularized configuration reaches $1.40\times$ speedup without increasing PPL. At the 3B scale, speedups are harder to realize with our current PyTorch-level implementation, but the IDN-regularized model still shows an encouraging frontier point: $1.20\times$ speedup with a 3.9% PPL increase.

This behavior is not simply exact numerical recovery of the sequential forward. Exact convergence of the Newton formulation recovers the sequential trace, but practical SNLP uses approximate structured corrections, finite iteration counts, chunking, fusion, and initialization choices. We therefore view IDN and HCN as inducing **solver-induced inference bias**: the finite-iteration computation differs from strict sequential execution in a way that can expose practical speed-quality tradeoffs and serve as a drafter in verified decoding. Our contributions are:

- We formulate layer-parallel language-model inference as structured surrogate Newton solving over the hidden-state trace, instantiated as IDN, HCN, and DiagN.
- We introduce SNLP-aware training, which can enable useful speed-quality tradeoffs with small PPL loss or mitigate downstream accuracy loss under SNLP inference.
- We introduce chunkwise layer fusion, which groups multiple depthwise-parallel layers into wider executable chunks before applying the structured Newton correction.
- We analyze the resulting solver-induced inference bias through correction ordering, propagation, variance-reduction, and layer-coupling ablations.

2 Related Work

Parallel nonlinear solvers. SNLP builds on the view that a sequential computation can be solved as a coupled nonlinear system. DEER applies Newton's method to nonlinear recurrences and uses

parallel scan to solve the resulting linearized dynamics (Lim et al., 2024); later work extends this perspective to MCMC chains (Zoltowski et al., 2025) and improves stability and scalability with quasi-Newton and Kalman-style approximations (Gonzalez et al., 2024). Song et al. (Song et al., 2021) frame feedforward computation as parallel nonlinear equation solving, and Jacobi decoding applies fixed-point iteration to parallelize autoregressive translation (Santilli et al., 2023). Deep Equilibrium Models (Bai et al., 2019) take a complementary view, finding fixed points of weight-tied infinite-depth networks via root-finding. Picard iterations parallelize diffusion-model sampling by exploiting identity Jacobian structure near Euler-discretized dynamics (Shih et al., 2023); (Gonzalez et al., 2026) unify Newton, Picard, and Jacobi under a linear dynamical systems framework and show convergence depends on Jacobian approximation fidelity. Our IDN is equivalent to Picard (identity approximation); DiagN and HCN are progressively richer quasi-Newton variants in this taxonomy. Our work rotates this line of work from sequence length to Transformer depth, and focuses on structured surrogates that avoid full layer Jacobians.

Associative scans and structured recurrences. Parallel prefix scan is a classical primitive (Blelloch, 1990) that has become central to efficient recurrent and state-space models. Linear recurrent networks can be parallelized over sequence length with scan (Martin & Cundy, 2018); structured state-space models such as S4 (Gu et al., 2022) and Mamba (Gu & Dao, 2024) use related hardware-aware recurrent kernels and scan-style algorithms. SNLP uses the same computational principle for depthwise correction: when the surrogate is identity, diagonal, or a small matrix, the Newton correction becomes a cheap structured recurrence.

Depth mixing and residual architectures. Residual connections (He et al., 2016) are central to deep Transformer training, and several architectures modify how information flows across depth. Hyper-Connections and mHC introduce learned residual-stream mixing and stabilization mechanisms (Zhu et al., 2025; Xie et al., 2025); AttnRes replaces fixed residual accumulation with learned attention over previous layer outputs (Team et al., 2026). Value residual learning and x0-style residual connections also alter how features persist through depth (Zhou et al., 2025; Jordan et al., 2024). Weight-tied and looped architectures, including Universal Transformers (Dehghani et al., 2019), ALBERT (Lan et al., 2020), recurrent-depth models (Geiping et al., 2025), and Hyperloop Transformers (Zeitoun et al., 2026), reuse layers across depth. SNLP is complementary: rather than only changing the forward architecture, it asks whether the resulting depth dynamics expose a cheap surrogate for Newton-style layer-parallel inference.

Efficient language-model inference. Most efficient LLM inference work accelerates token-level decoding through batching, KV caching (Kwon et al., 2023), quantization, memory-aware execution (Alizadeh et al., 2024), kernel engineering (Dao et al., 2022), speculative decoding (Leviathan et al., 2023; Chen et al., 2023), early exit (Schuster et al., 2022), or serving systems (Zhen et al., 2025; Miao et al., 2025). These techniques improve the execution of the standard sequential layer stack, whereas SNLP targets a different bottleneck: the dependency chain across layers for a fixed token prefix. Our experiments use Nanochat as a compact from-scratch training and evaluation harness (Karpathy, 2025); we also run preliminary post-hoc and finetuning experiments on representative open-weight decoder-only models, including Qwen2.5 and TinyLlama (Team, 2024; Zhang et al., 2024). The gap between trained-from-scratch and off-the-shelf results suggests that layer-parallel inference benefits from training/inference co-design, leaving stronger pretrained-model adaptation to future work.

Layer-parallel computation. (Jiang et al., 2026) apply multigrid-in-time to parallelize Transformer *training* across layers via a neural ODE formulation, complementing our inference-side approach. (Calvo González et al., 2026) parallelize consecutive layer pairs post hoc by dropping inter-layer dependencies; SNLP instead applies Newton corrections and co-designs training with inference.

3 METHODS

3.1 BACKGROUND

Layer traces as residual equations. Consider a depth-L model with hidden states $h_0, \dots, h_L$

$$h_l = f_l(h_{l-1}), \qquad l = 1, \dots, L. \tag{1}$$

Here l indexes depth, while superscripts such as (k) will index iterative solver steps. Rather than viewing the forward pass only as a sequential program, we can view the entire hidden-state trace

$\mathbf{h} = (h_1, \ldots, h_L)$ as the solution of a nonlinear residual equation. Define

$$G_l(\mathbf{h}) = h_l - f_l(h_{l-1}), \qquad G(\mathbf{h}) = (G_1(\mathbf{h}), \ldots, G_L(\mathbf{h})). \tag{2}$$

The usual sequential forward pass is exactly the zero-residual trace $G(\mathbf{h}) = 0$. This formulation exposes a different source of parallelism: instead of computing layers one after another, one may iteratively solve for all layer states jointly.

Newton-style updates over depth. DEER applies Newton's method to nonlinear recurrences by linearizing the transition at the current iterate and solving the resulting linear recurrence in parallel (Lim et al., 2024; Zoltowski et al., 2025). Rotating this view by $90°$, the depth axis of any block-sequential model–a Transformer, CNN, or recurrent stack–can be treated as the recurrence axis. At solver iteration k, the exact Newton update over layers can be written as

$$h_l^{(k+1)} = f_l\left(h_{l-1}^{(k)}\right) + J_l^{(k)}\left(h_{l-1}^{(k+1)} - h_{l-1}^{(k)}\right), \qquad J_l^{(k)} = \tfrac{\partial f_l}{\partial h_{l-1}}\left(h_{l-1}^{(k)}\right). \tag{3}$$

This recurrence is equivalent to applying Newton's method to the stacked residual system in Eqn. 2 because the residual Jacobian is block lower-bidiagonal; we refer readers to prior derivations of this equivalence in DEER-style solvers (Zoltowski et al., 2025; Gonzalez et al., 2024). The challenge is that $J_l^{(k)}$ is the Jacobian of an entire layer or block output with respect to its input. For language-model hidden states, materializing this operator is infeasible, and even Jacobian-vector products or finite-difference approximations can consume the latency budget that layer parallelism is meant to save. Naive fixed-point updates avoid this cost but are often unstable on trained residual networks. The practical question is therefore whether we can replace the exact layer Jacobian with a cheap structured surrogate that preserves enough of the Newton correction to make finite-iteration, layer-parallel inference useful.

3.2 Structured Newton Layer Parallelism

SNLP replaces the exact layer Jacobian in Eqn. 2 with a cheap structured surrogate. Let the first S layers be evaluated sequentially, producing a prefix state h_S. The remaining suffix $\{S+1, \ldots, L\}$ is solved by iterative correction. At iteration k, each suffix layer is first evaluated using the current estimate of its input,

$$\widetilde{h}_l^{(k)} = f_l\left(h_{l-1}^{(k)}\right), \qquad l = S+1, \ldots, L. \tag{4}$$

These evaluations are independent across l and can be batched or fused. SNLP then applies the structured Newton correction

$$h_l^{(k+1)} = \widetilde{h}_l^{(k)} + A_l^{(k)}\left(h_{l-1}^{(k+1)} - h_{l-1}^{(k)}\right), \qquad h_S^{(k+1)} = h_S, \tag{5}$$

where $A_l^{(k)}$ is a surrogate for the exact block Jacobian $J_l^{(k)}$. If $A_l^{(k)} = J_l^{(k)}$, this recovers the exact DEER/Newton update over depth. SNLP instead chooses $A_l^{(k)}$ so that the correction is much cheaper than evaluating or materializing the true Jacobian, while still propagating information from earlier corrected layer states to later ones.

The update in Eqn. 5 separates the two costs that matter for inference. The nonlinear layer evaluations $\widetilde{h}_l^{(k)}$ are parallel across the suffix and dominate GPU work. The Newton correction still propagates through depth, but for IDN/HCN the surrogate is trivial to compute or available from the architecture, making correction cheap relative to a Transformer block. Thus SNLP realizes speedup by parallelizing the expensive block forwards while keeping only a lightweight structured recurrence on the critical path. After K iterations, the model projects the final corrected state $h_L^{(K)}$ to logits.

Effect of the correction. The correction in Eqn. 5 is what moves information across the whole suffix within a single solver iteration. Once the layer outputs $\widetilde{h}_l^{(k)}$ are computed, the corrected prefix state propagates from layer S to layer L through the structured recurrence, so $h_L^{(k+1)}$ depends on the corrections from all layers $S+1, \ldots, L$. Without this correction, a naive parallel fixed-point update only advances information by one layer per iteration: after K iterations, the effect of the prefix can reach only the next K layers of the suffix. We verify this propagation effect empirically in Section 5.3 and Section E.8.

3.3 Structured Surrogates

Identity Newton (IDN). For residual Transformer blocks, $f_l(x)$ contains an explicit identity path. SNLP uses the architecture-induced surrogate

$$A_l^{(k)} = I. \tag{6}$$

The correction becomes

$$h_l^{(k+1)} = \widetilde{h}_l^{(k)} + h_{l-1}^{(k+1)} - h_{l-1}^{(k)}, \tag{7}$$

which reduces the Newton correction to additive propagation of the previous-layer correction. This is our main residual-Transformer instantiation because it requires no Jacobian estimation and makes the correction essentially a prefix-sum over depth. We refer to this variant as Identity Newton (IDN).

HC Newton (HCN). For hyper-connection and mHC-style models (Zhu et al., 2025; Xie et al., 2025), the architecture exposes an explicit residual mixing matrix over streams. If a block applies residual mixing matrices $H^{\text{res}}_{\text{attn},l}$ and $H^{\text{res}}_{\text{mlp},l}$, we use

$$A_l = H^{\text{res}}_{\text{mlp},l} H^{\text{res}}_{\text{attn},l}. \tag{8}$$

This surrogate is small: it acts on the stream dimension rather than on the full hidden dimension. The mHC case demonstrates that SNLP is not tied to the identity residual path; any architecture with a cheap structured approximation to inter-layer sensitivity can define an SNLP correction.

Diagonal Newton (DiagN). A closer approximation to the exact Newton step uses only the diagonal of the layer Jacobian,

$$A_l^{(k)} = \operatorname{diag}\left(J_l^{(k)}\right). \tag{9}$$

This connects SNLP to quasi-DEER and ELK-style approximations (Gonzalez et al., 2024). With a diagonal surrogate, the correction in Eqn. 5 becomes an elementwise affine recurrence over depth and can be evaluated efficiently by an associative prefix scan (Blelloch, 1990; Martin & Cundy, 2018; Gu & Dao, 2024). In our implementation, the diagonal can be estimated by a Hutchinson-style finite-difference or VJP estimator (Hutchinson, 1990; Zoltowski et al., 2025; Bekas et al., 2007), optionally only on a subset of layers.

3.4 SNLP-Aware Training

Off-the-shelf sequential models need not have layer dynamics that match a cheap surrogate. We use SNLP-aware training in two ways: an auxiliary matching loss during pretraining, and direct SNLP-forward SFT during task adaptation. For pretraining, we add a loss that makes a finite SNLP solve match the sequential trace. For each suffix length $N \in \mathcal{S}$, let $\mathcal{T}_N$ be the stride-selected supervised layers in that suffix, and let $\widehat{h}_l^{\text{SNLP}}(N, K; A)$ be the SNLP state at layer l after K iterations with surrogate family A. We optimize

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda \sum_{N \in \mathcal{S}} \sum_{l \in \mathcal{T}_N} \frac{\left\| \widehat{h}_l^{\text{SNLP}}(N,K;A) - h_l^{\text{seq}} \right\|_2}{\left\| h_l^{\text{seq}} \right\|_2 + \epsilon}. \tag{10}$$

In our runs, $K = 1$ during training and $\mathcal{S}$ contains one or more configured suffix lengths. The set $\mathcal{T}_N$ controls where the matching loss is applied: stride 0 uses only the final layer, $\mathcal{T}_N = \{L\}$, while positive strides add sparse intermediate layers and always include L to reduce memory cost; see Table 7 for ablations. The surrogate A is identity for IDN, the stream-mixing matrix for HCN, and diagonal for DiagN. This objective does not make layers removable; rather, it makes the chosen structured correction a better finite-iteration solver for the sequential trace. For SNLP-forward SFT, we instead use the SNLP computation itself as the model forward pass and optimize the standard supervised loss, allowing task adaptation to the approximate layer-parallel computation used at inference.

3.5 Inference With Fusion and Chunking

At inference time, SNLP runs a sequential prefix and applies Eqn. 5 to a suffix of $N = L - S$ layers. The suffix hidden states can be initialized from the prefix state h_S, from a one-shot batched

forward, or from a lightweight predictor; our main evaluations focus on simple prefix-state and batched-forward initializations. The number of iterations K controls the quality-cost tradeoff.

Layer fusion. Wall-clock speedups require more than replacing the Jacobian. We therefore combine SNLP correction with GPU-oriented execution of the suffix. In the batched form, per-layer weights are stacked so all suffix layers evaluate in one grouped operation. In the fused form, several layers that read the same input are combined into one wider layer: the attention Q, K, V projections and MLP expansion matrices are concatenated along their output dimension, while the attention output projection and MLP down-projection are concatenated along their input dimension. Equivalently, the fused layer computes all branch outputs in one wide matmul and performs the required sum-reductions after the attention output projection and after the MLP projection. This converts layer-parallel algorithmic structure into larger GPU-efficient matrix multiplies.

Chunkwise strategy. For more aggressive parallelization, we split the suffix into multiple fused chunks, inspired by DeltaNet-style chunkwise parallelization (Yang et al., 2024). Each chunk is treated as a wide layer as above, and all chunk forwards are parallelizable because they use the current chunk-input estimates from iteration k. SNLP then applies the structured Newton correction between chunk outputs rather than between individual layers:

$$h_c^{(k+1)} = \widetilde{h}_c^{(k)} + A_c^{(k)} \left(h_{c-1}^{(k+1)} - h_{c-1}^{(k)} \right), \tag{11}$$

where c indexes chunks and $A_c^{(k)}$ is the corresponding identity or architecture-induced chunk surrogate. Chunking trades a coarser solver approximation for better hardware utilization, since the expensive work is executed as a small number of wide parallel chunk forwards followed by a cheap correction across chunks. These fusion choices change the finite-iteration computation, so the resulting model should be understood as practical SNLP inference rather than exact recovery of the sequential forward.

4 Analysis

Exact convergence of Newton's method on Eqn. 2 recovers the sequential forward pass, but practical SNLP uses approximate surrogates, finite iterations, initialization, fusion, and chunking; together these define a solver-induced inference bias. We summarize the main mechanisms here and defer derivations to the appendix.

Training-side effects. SNLP-aware training encourages a cheap structured correction to match the sequential final state. For residual blocks $f_l(x) = x + g_l(x)$, IDN training encourages $g_l(h_S) \approx g_l(h_{l-1}^{\text{seq}})$ over the suffix, putting implicit Lipschitz pressure on the non-residual branch: smaller J_{g_l} makes $J_{f_l} = I + J_{g_l}$ closer to the IDN surrogate. This encourages suffix layers to act more like stable feature-correction modules. The suffix is still useful, but its features become less sensitive to the exact state where they are evaluated.

Inference-side effects. IDN and fused SNLP evaluate many suffix contributions at a common or chunk-level input instead of along the fully accumulated sequential chain. This can reduce layerwise error compounding under the approximation in Section B.1, at the cost of bias from evaluating $g_l(h_S)$ rather than $g_l(h_{l-1}^{\text{seq}})$. When SNLP training keeps this bias small, this offers a possible explanation for remaining near the sequential model while exposing latency tradeoffs. Fusion and chunking add another bias: fused branches see summed cross-layer signals, which can alter feature interactions and can hurt when chunks are too aggressive.

Layer coupling. The results suggest that SNLP benefits from structured depth coupling rather than removing depth interactions entirely. HC and mHC parameterize residual-stream mixing (Zhu et al., 2025; Xie et al., 2025), while AttnRes learns attention over depth (Team et al., 2026). SNLP is complementary: IDN uses identity coupling, HCN uses the learned residual mixing matrix, and fused SNLP induces implicit cross-layer coupling inside each chunk.

5 Experiments

From-scratch models. Our main results use Nanochat (Karpathy, 2025) models trained from scratch at two 32-layer scales: a 3B model with $n_{\text{embd}} = 2048$ and 16 heads, and a 0.5B model with $n_{\text{embd}} = 640$ and 5 heads. Forward Nanochat uses rotary position embeddings (Su et al., 2024), an

Model	Config	K	PPL	ΔPPL	Speedup	Top-1	LogitSim	EmbSim	AR Match
Nanochat-3B standard									
No Reg. [10.10]	`8xF1-h0`	4	10.65	+5.5%	0.95×	85.8%	0.993	0.965	61.7%
	`8xF1-h0`	8	10.13	+0.3%	0.77×	99.4%	1.000	0.996	95.9%
IDN Reg. [10.07 (-0.3%)]	`8xF1-h0`	1	10.46	+3.9%	1.20×	84.0%	0.985	0.961	55.0%
	`8xF1-fwd`	1	10.43	+3.6%	1.10×	85.0%	0.988	0.963	58.0%
	`8xF1-h0`	4	10.09	+0.2%	0.95×	99.1%	1.000	0.995	95.2%
Nanochat-0.5B standard									
No Reg. [15.21]	`8xF1-h0`	4	15.42	+1.4%	0.99×	95.2%	1.000	0.993	90.1%
	`8xF1-fwd`	4	15.35	+0.9%	0.93×	97.8%	1.000	0.997	95.1%
IDN Reg. [15.36 (+1.0%)]	`24xF1-h0`	2	18.09	+17.8%	1.88×	74.3%	0.989	0.969	54.5%
	`24xF1-h0`	4	16.42	+6.9%	1.33×	90.0%	0.998	0.987	81.4%
	`12xF1-fwd`	2	15.59	+1.5%	1.14×	95.0%	0.999	0.993	90.8%
Nanochat-0.5B w/o x0ve									
No Reg. [17.65]	`20xF1-fwd`	4	18.91	+7.2%	1.27×	90.3%	0.999	0.989	86.1%
	`8xF1-h0`	4	17.75	+0.6%	0.98×	99.0%	1.000	0.999	98.4%
IDN Reg. [17.57 (-0.5%)]	`12xF2-h0`	1	20.55	+17.0%	2.58×	70.5%	0.996	0.955	56.4%
	`12xF2-h0`	2	18.46	+5.1%	2.09×	79.4%	0.998	0.966	80.2%
	`1xF12-h0`	1	17.56	-0.1%	1.40×	87.7%	0.999	0.976	100.0%
Nanochat-0.5B-mHC									
No Reg. [15.16]	`8xF1-fwd`	2	15.65	+3.2%	1.10×	93.4%	0.998	0.991	90.9%
	`8xF1-h0`	4	15.38	+1.5%	1.00×	98.2%	1.000	0.997	95.9%
HCN Reg. [15.52 (+2.4%)]	`16xF1-h0`	1	16.93	+9.1%	1.60×	81.4%	0.993	0.980	72.0%
	`16xF1-h0`	4	15.71	+1.2%	1.14×	97.3%	1.000	0.995	94.4%
	`8xF1-h0`	4	15.60	+0.5%	1.00×	98.3%	1.000	0.997	95.9%

Table 1: Main results for layer-parallel decoding across Nanochat model variants using 2048-token PPL evaluation. Brackets under model names give sequential PPL and, when shown, the change from the corresponding No Reg. baseline. Entries marked gray denote comparable quality or speed: PPL increase within 2.5%, or speedup at least 0.95× but below sequential. Speed-oriented configurations reach up to 2.58× practical speedup, while quality-oriented configurations recover near-sequential PPL with high LogitSim, EmbSim, and AR Match. AR Match uses 32 generated tokens, so 50% observed match requires roughly 95% conditional per-token agreement (Section E.1).

x_0 residual connection (Jordan et al., 2024), and value embeddings (Zhou et al., 2025). For both standard scales, we train No Reg., IDN Reg., and DiagN Reg. variants. We also train 0.5B variants without x_0/VE to isolate SNLP from Nanochat-specific features, and an mHC model (Zhu et al., 2025; Xie et al., 2025), where HCN uses the learned matrix surrogate.

Off-the-shelf models. We also evaluate SNLP post hoc on Qwen2.5-0.5B-Instruct (Team, 2024) and TinyLlama-1.1B-Chat-v1.0 (Zhang et al., 2024). Their best SNLP configurations can match sequential perplexity and are slower than sequential execution; see Table 14. We additionally run a preliminary TinyLlama finetuning study to test whether SNLP compatibility can be introduced after pretraining.

Inference configurations. We evaluate chunkwise SNLP with IDN correction for residual models and HCN correction for mHC models. A configuration `NxFM-init` denotes N parallel chunks, each fusing M layers, with initialization `h0` from the prefix state h_S or `fwd` from a one-shot parallel forward $f_l(h_S)$. For example, `8xF2-fwd` uses 8 fused 2-layer chunks with one-shot initialization. Preheat initialization is deferred to the appendix.

Metrics and protocol. All models are implemented in PyTorch (Paszke et al., 2019) using HuggingFace Transformers (Wolf et al., 2020). Training and evaluation use ClimbMix dataset (Diao et al., 2025); PPL is measured on a fixed 1M-token validation split. Timing uses batch size 1 on H100 GPUs with 50 warmup and 200 measured runs. Top-1 and LogitSim are prefill metrics against the original sequential model. AR Match is token-level greedy-generation agreement over 32 tokens; for fused-weight variants, it compares to sequential execution of the same fused model. EmbSim compares generated text to original sequential generation using BGE-small-en-v1.5 embeddings (Xiao et al., 2023).

	IDN Reg.						No Reg.					
Layer	$\sigma_{\max}$	$\|J\|_F$	amp.	$10^3\|\epsilon_l\|$	$10^3 C_\epsilon$	Δ_g	$\sigma_{\max}$	$\|J\|_F$	amp.	$10^3\|\epsilon_l\|$	$10^3 C_\epsilon$	Δ_g
25	1.66	2.40	2.92	0.271	0.271	0.056	18.25	24.75	16.57	20.463	20.463	0.159
27	1.75	2.54	2.79	0.458	0.827	0.142	21.93	27.01	14.91	46.101	61.991	0.272
29	1.69	2.45	2.55	1.453	2.379	0.277	19.87	25.66	12.29	66.422	123.962	0.436
31	1.97	2.84	2.73	1.250	3.895	0.269	30.26	34.44	13.43	239.148	351.378	0.717

Table 2: Per-layer Jacobian, amplification, and substitution-error diagnostics on selected suffix layers of 0.5B standard models. The $10^3|\epsilon_l|$ and $10^3 C_\epsilon$ columns report relative substitution errors $\times 1000$.

	Sequential				IDN ($K = 1$)				
N	Forward	Reversed	Best shuffle	Shuffle std.	Forward	Reversed	Best shuffle	No correction	Shuffle std.
8	**16.8**	16.9	16.8	0.05	**17.2**	37.4	17.6	37.4	6.63
12	**16.8**	17.2	16.9	0.06	17.9	67.2	**17.8**	67.2	25.62
16	**16.8**	316.4	17.0	2793.71	**18.0**	div.	30.0	div.	565.01

Table 3: Correction-ordering summary for the 0.5B IDN Reg. model at sequence length 2048. Best shuffle is the lowest PPL among random shuffles; shuffle std. is computed over finite random-shuffle PPL values; `div.` denotes divergence. All IDN runs use `h0` initialization.

5.1 Main results

Pretraining evaluation. Table 1 reports two SNLP configurations for each base model and three for each SNLP-regularized model; checkpoint settings are listed in Table 6. Rows are ordered from higher to lower PPL within each variant: aggressive parallelism favors speed, while additional iterations recover quality at higher cost. On the small 0.5B w/o x_0/VE IDN-regularized model, SNLP reaches $1.40\times$ speedup without increasing PPL, or up to $2.58\times$ speedup with 17.0% PPL increase. The weaker 3B gains are consistent with less spare parallel capacity and a smaller parallelizable suffix. Profiling identifies prefix and correction overheads, while an FHE-oriented simulation illustrates a different depth-sensitive cost regime (Appendix F). Custom fused kernels or software-hardware co-design, such as compute-in-memory-style execution (Wan et al., 2022), may be needed to realize the algorithmic parallelism at larger scale.

The diagonal variant is useful as an ablation because it relaxes the strong identity assumption, but it adds extra block evaluations or autodiff work. Table 15 shows that, unlike IDN inference, most quality-oriented diagonal-correction configurations only recover comparable PPL to sequential inference. Compared with IDN, DiagN introduces a different solver-induced bias that is numerically closer to the sequential computation, but this does not translate into a better practical speed-quality frontier in our implementation.

Downstream task evaluation. IDN regularization mitigates, but does not eliminate, downstream accuracy loss. On 3B after sequential SFT, IDN inference reaches 57.03% versus 54.60% without regularization and 57.85% for the base sequential pipeline. Full comparisons are in Appendix D.1.

Self-speculative decoding. SNLP-only drafters reach $1.67\times$ estimated end-to-end speedup with sequential verification preserving greedy outputs (Appendix D.2). Combining MTP with IDN improves efficiency further in a separate matched comparison (Appendix D.3).

5.2 Training effect

Moderate IDN/HCN regularization usually adds a small amount of pretraining PPL loss, with sequential PPL changes ranging from -0.5% to +2.4% in Table 1. However, as shown in the SFT results in Appendix D.1, IDN-regularized models can maintain their parallel behavior after standard SFT. The diagnostics and inference-effect ablations in this section use a more parallelizable 0.5B IDN checkpoint from Table 7 ($N = 24$, $\lambda = 0.5$, stride 3, no detach; PPL 16.81), rather than the main selected 0.5B IDN checkpoint. Table 2 reports non-residual-branch Jacobian diagnostics for selected suffix layers of the 0.5B standard models. IDN Reg. reduces spectral estimates by roughly 11–$15\times$ and Frobenius estimates by 10–$12\times$, supporting the implicit Lipschitz-regularization interpretation. Full-layer amplification $\|J_{f_l} v\|/\|v\|$ is also lower at every reported layer (Table 2). Diagonal correction adds estimation cost and can be unstable; see Appendix E.5.

5.3 INFERENCE EFFECT

Correction ordering. Table 3 tests whether correction order matters on the IDN Reg. model; full tables are in Section E.7. Sequentially executing the permuted suffix is almost invariant to layer order up to $N = 12$, with very low shuffle variance, suggesting that IDN regularization makes these suffix layers effectively parallelizable. SNLP with IDN correction is more order-sensitive: the best $K = 1$ PPL in the summary is almost always achieved by forward order, possibly because the correction recurrence preserves the causal direction of the original depth computation.

Correction propagation. Layer-activation ablations in Section E.8 confirm the propagation pattern from Section 3: with correction, every active suffix layer can affect the final output in one iteration; without correction, influence moves only one layer per iteration, producing a staircase pattern.

Variance reduction. For each suffix layer l, write $f_l(h) = x_{\mathrm{in},l}(h) + g_l(h)$, where g_l is the non-residual attention/MLP update. Let h_S be the clean prefix state and h_{l-1}^{seq} the sequential input to layer l. We define the substitution error $\epsilon_l = g_l(h_{l-1}^{\mathrm{seq}}) - g_l(h_S)$ and report:

$$|\epsilon_l| = \frac{\mathbb{E}_b[\|\epsilon_{l,b}\|_2]}{\mathbb{E}_b[\|h_{S,b}\|_2]+\epsilon}, \qquad C_\epsilon(l) = \frac{\mathbb{E}_b\left[\left\|\sum_{r=S+1}^{l} \epsilon_{r,b}\right\|_2\right]}{\mathbb{E}_b[\|h_{S,b}\|_2]+\epsilon}, \qquad \Delta_g(l) = \frac{\mathbb{E}_b\left[\left\|g_l(h_{l-1,b}^{\mathrm{seq}}) - g_l(h_{S,b})\right\|_2\right]}{\mathbb{E}_b\left[\|g_l(h_{S,b})\|_2\right]+\epsilon}. \tag{12}$$

Here $|\epsilon_l|$ measures the per-layer substitution error relative to the prefix-state scale, $C_\epsilon(l)$ measures how these errors accumulate through the suffix, and $\Delta_g(l)$ measures sensitivity relative to the layer update magnitude. Table 2 shows that IDN Reg. keeps relative per-layer error in the 0.03%–0.15% range of $\|h_S\|$, while No Reg. ranges from 2% to 24%; IDN Reg. also has lower Δ_g than No Reg. at every reported layer.

N	Seq layers	IDN Reg.	No Reg.
0	32	16.82	15.21
8	24	50.61	60.29
12	20	156.67	285.32
16	16	1785.02	4295.03

Table 4: Early-exit PPL ($K = 0$) for the 0.5B models at sequence length 2048. $N = 0$ is ordinary sequential inference.

What $J \approx I$ really means. Input-invariant suffix layers are not droppable. For the IDN Reg. ablation checkpoint, skipping 8–16 suffix layers greatly increases PPL (Table 4). Thus $J \approx I$ means that the non-residual features are nearly input-invariant, not unnecessary.

6 CONCLUSION

We introduced Structured Newton Layer Parallelism, a training and inference framework for relaxing the strict layerwise dependency in Transformer inference. SNLP treats the hidden-state trace across depth as a residual equation, but replaces exact layer Jacobians with cheap structured surrogates induced by the architecture or training objective. In residual Transformers this yields IDN, while mHC-style models use HCN with a learned matrix surrogate. These structured corrections make layer-parallel decoding practical enough to expose a useful speed-quality frontier: on 0.5B Nanochat models, SNLP reaches up to $2.58\times$ wall-clock speedup, and a less aggressive configuration reaches $1.40\times$ speedup without increasing PPL. Finite-iteration IDN and HCN introduce solver-induced inference bias, exposing latency–quality tradeoffs without requiring exact recovery of the sequential trace. SNLP-aware training also mitigates downstream accuracy loss, and SNLP drafters support self-speculative decoding with sequential verification.

Discussion. SNLP can trade additional compute and activation memory for a shorter sequential critical path, so latency gains depend on hardware utilization and parallelizable depth. The strongest measured gains are at 0.5B; the 3B IDN-regularized model reaches $1.20\times$ speedup with 3.9% higher PPL. Downstream accuracy gaps and adaptation to off-the-shelf models remain challenges. Future work includes stronger post-training adaptation, systematic SFT studies, and optimized kernels and hardware co-design for larger models. Further directions include cheaper diagonal corrections, composition with token-parallel decoding, extension to diffusion language models, and encrypted execution to test the benefits suggested by our FHE simulation.

REFERENCES

Keivan Alizadeh, Seyed Iman Mirzadeh, Dmitry Belenko, S Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. Llm in a flash: Efficient large language model inference with limited memory. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12562–12584, 2024.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. Smollm2: When smol goes big – data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32:688–699, 2019.

Costas Bekas, Effrosyni Kokiopoulou, and Yousef Saad. An estimator for the diagonal of a matrix. *Applied numerical mathematics*, 57(11-12):1214–1229, 2007.

Guy E Blelloch. Prefix sums and their applications. In *Synthesis of Parallel Algorithms*, pp. 35–60. Morgan Kaufmann, 1990.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Ramón Calvo González, Daniele Paliotta, Matteo Pagliardini, Martin Jaggi, and François Fleuret. Leveraging the true depth of LLMs. *Transactions on Machine Learning Research*, 2026.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. Cryptology ePrint Archive, Paper 2016/421, 2016. URL https://eprint.iacr.org/2016/421.

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.

Federico Danieli, Miguel Sarabia, Xavier Suau Cuadros, Pau Rodriguez, and Luca Zappella. Deeppcr: Parallelizing sequential operations in neural networks. *Advances in Neural Information Processing Systems*, 36:47598–47625, 2023.

Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35, 2022.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan Su, Markus Kliegl, Zijia Chen, Peter Belcak, Yoshi Suhara, Hongxu Yin, Mostofa Patwary, Celine Lin, Jan Kautz, and Pavlo Molchanov. Nemotron-climb: Clustering-based iterative data mixture bootstrapping for language model pre-training. *Advances in Neural Information Processing Systems*, 38, 2025.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Advances in Neural Information Processing Systems*, 38, 2025.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024. URL https://arxiv.org/abs/2404.19737.

Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. Diffucoder: Understanding and improving masked diffusion models for code generation. *arXiv preprint arXiv:2506.20639*, 2025.

Xavier Gonzalez, Andrew Warrington, Jimmy TH Smith, and Scott W Linderman. Towards scalable and stable parallelization of nonlinear rnns. *Advances in Neural Information Processing Systems*, 37:5817–5849, 2024.

Xavier Gonzalez, E. Kelly Buchanan, Hyun Dong Lee, Jerry Weihong Liu, Ke Alexander Wang, David M. Zoltowski, Leo Kozachkov, Christopher Ré, and Scott W. Linderman. A unifying framework for parallelizing sequential models with linear dynamical systems. *Transactions on Machine Learning Research*, 2026.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Xu Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in Neural Information Processing Systems*, 32:103–112, 2019.

Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics – Simulation and Computation*, 19(2):433–450, 1990.

Shuai Jiang, Marc Salvadó-Benasco, Eric C. Cyr, Alena Kopaničáková, Rolf Krause, and Jacob B. Schroder. Layer-parallel training for transformers. *arXiv preprint arXiv:2601.09026*, 2026.

Keller Jordan, Jeremy Bernstein, Brendan Rappazzo, Boza Vlado, You Jiacheng, Franz Cesista, and Braden Koszarsky. modded-nanogpt: Speedrunning the nanogpt baseline, 2024. URL https://github.com/KellerJordan/modded-nanogpt.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Andrej Karpathy. nanochat: The best chatgpt that $100 can buy, 2025. URL https://github.com/karpathy/nanochat.

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 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Yi Heng Lim, Qi Zhu, Joshua Selfridge, and Muhammad Firmansyah Kasim. Parallelizing non-linear sequential models over the sequence length. In *The Twelfth International Conference on Learning Representations*, 2024.

Eric Martin and Chris Cundy. Parallelizing linear recurrent neural nets over sequence length. In *International Conference on Learning Representations*, 2018.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Hongyi Jin, Tianqi Chen, and Zhihao Jia. Towards efficient generative large language model serving: A survey from algorithms to systems. *ACM Computing Surveys*, 58(1):1–37, 2025.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32: 8024–8035, 2019.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Andrea Santilli, Silvio Severino, Emilian Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodola. Accelerating transformer inference for translation via parallel decoding. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12336–12355, 2023.

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

Andy Shih, Suneel Belkhale, Stefano Ermon, Dorsa Sadigh, and Nima Anari. Parallel sampling of diffusion models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Yang Song, Chenlin Meng, Renjie Liao, and Stefano Ermon. Accelerating feedforward computation via parallel nonlinear equation solving. In *International Conference on Machine Learning*, pp. 9791–9800. PMLR, 2021.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Muffin, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.

Kimi Team, Guangyu Chen, Yu Zhang, Jianlin Su, Weixin Xu, Siyuan Pan, Yaoyu Wang, Yucheng Wang, Guanduo Chen, Bohong Yin, Yutian Chen, Junjie Yan, Ming Wei, Y. Zhang, Fanqing Meng, Chao Hong, Xiaotong Xie, Shaowei Liu, Enzhe Lu, Yunpeng Tai, Yanru Chen, Xin Men, Haiqing Guo, Y. Charles, Haoyu Lu, Lin Sui, Jinguo Zhu, Zaida Zhou, Weiran He, Weixiao Huang, Xinran Xu, Yuzhi Wang, Guokun Lai, Yulun Du, Yuxin Wu, Zhilin Yang, and Xinyu Zhou. Attention residuals, 2026.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

Yao Teng, Han Shi, Xian Liu, Xuefei Ning, Guohao Dai, Yu Wang, Zhenguo Li, and Xihui Liu. Accelerating auto-regressive text-to-image generation with training-free speculative jacobi decoding. In *International Conference on Learning Representations*, 2025.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30:5998–6008, 2017.

Weier Wan, Rajkumar Kubendran, Clemens Schaefer, Sukru Burc Eryilmaz, Wenqiang Zhang, Dabin Wu, Stephen Deiss, Priyanka Raina, He Qian, Bin Gao, Siddharth Joshi, Huaqiang Wu, and H-S Philip Wong. A compute-in-memory chip based on resistive random-access memory. *Nature*, 608(7923):504–512, 2022.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, 2020.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. C-pack: Packaged resources to advance general chinese embedding, 2023.

Zhenda Xie, Yixuan Wei, Huanqi Cao, Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Jiang Chang, Kuai Yu, et al. mhc: Manifold-constrained hyper-connections. *arXiv preprint arXiv:2512.24880*, 2025.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *Advances in Neural Information Processing Systems*, 37, 2024.

Abbas Zeitoun, Lucas Torroba-Hennigen, and Yoon Kim. Hyperloop transformers. *arXiv preprint arXiv:2604.21254*, 2026.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024.

Ranran Zhen, Juntao Li, Yixin Ji, Zhenlin Yang, Tong Liu, Qingrong Xia, Xinyu Duan, Zhefeng Wang, Baoxing Huai, and Min Zhang. Taming the titans: A survey of efficient llm inference serving. *arXiv preprint arXiv:2504.19720*, 2025.

Zhanchao Zhou, Tianyi Wu, Zhiyun Jiang, Fares Obeid, and Zhenzhong Lan. Value residual learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28341–28356, 2025.

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-connections. In *The Thirteenth International Conference on Learning Representations*, 2025.

David M Zoltowski, Skyler Wu, Xavier Gonzalez, Leo Kozachkov, and Scott W Linderman. Parallelizing mcmc across the sequence length. *Advances in Neural Information Processing Systems*, 38, 2025.

APPENDIX

A Algorithm . . . 14
B Analysis Details . . . 15
B.1 Variance Reduction Derivation . . . 15
B.2 Connection to Hyper-Connections and mHC . . . 16
B.3 Fused Cross-Layer Coupling . . . 16
C Training Configuration and Ablations . . . 17
D Downstream Evaluation and Self-Speculative Decoding . . . 18
D.1 Downstream Task Evaluation . . . 18
D.2 Self-Speculative Decoding . . . 19
D.3 Comparison with Multi-Token Prediction . . . 20
E Additional Inference Ablations . . . 20
E.1 Interpreting AR Match Rate . . . 20
E.2 Coupling SNLP with Jacobi Decoding . . . 21
E.3 ELK Tempering and Preheat Initialization . . . 22
E.4 Off-the-Shelf Model Results . . . 22
E.5 Diagonal-Jacobian Correction Results . . . 22
E.6 Pretrained TinyLlama Fine-Tuning . . . 23
E.7 Correction Ordering . . . 24
E.8 Correction Propagation . . . 24
F Efficiency Regime and Additional Measurements . . . 25
F.1 Component Profiling and Activation Memory . . . 27
F.2 FHE-Compatible Approximation Simulation . . . 27

A ALGORITHM

The main SNLP update is given in Eqn. 5. Algorithm 1 shows the standard layer-by-layer execution, while Algorithm 2 shows the batched structured-correction implementation used by the `NxF1-h0` configurations. For HCN, A_l acts only on the HC/mHC stream dimension rather than the full hidden dimension.

Algorithm 1 Standard Sequential Layer Execution

Require: input tokens x, Transformer blocks $f_1, \ldots, f_L$
1: $h_0 \leftarrow \mathrm{Embed}(x)$
2: **for** $l = 1, \ldots, L$ **do**
3: $\quad h_l \leftarrow f_l(h_{l-1})$
4: **end for**
5: **return** $\mathrm{LMHead}(h_L)$

In Algorithm 3, FD, JVP, and VJP change only how the diagonal estimator is obtained; once $a_j = \mathrm{diag}(J^{(k)}_{S+j})$ is available, all variants use the same affine scan recurrence.

Algorithm 2 IDN/HCN Batched SNLP Inference with `h0` Initialization

Require: input tokens x, prefix length S, suffix length $N = L - S$, iterations K

$h_0 \leftarrow \mathrm{Embed}(x)$
for $l = 1, \ldots, S$ **do**
  $h_l \leftarrow f_l(h_{l-1})$
end for
$h_S^{(0)} \leftarrow h_S$; $h_{S+j}^{(0)} \leftarrow h_S$ for $j = 1, \ldots, N$
for $k = 0, \ldots, K-1$ **do**
  parallel for $j = 1, \ldots, N$: $\widetilde{h}_{S+j}^{(k)} \leftarrow f_{S+j}\Big(h_{S+j-1}^{(k)}\Big)$
  $h_S^{(k+1)} \leftarrow h_S$
  for $j = 1, \ldots, N$ **do**
    $h_{S+j}^{(k+1)} \leftarrow \widetilde{h}_{S+j}^{(k)} + A_{S+j}^{(k)}\Big(h_{S+j-1}^{(k+1)} - h_{S+j-1}^{(k)}\Big)$ {$A = I$ for IDN; Eqn. 8 for HCN.}
  end for
end for
return $\mathrm{LMHead}(h_L^{(K)})$

Algorithm 3 DiagN Inference with Associative Scan

Require: input tokens x, prefix length S, suffix length $N = L - S$, iterations K

Run the sequential prefix to obtain h_S; initialize $h_{S+j}^{(0)} \leftarrow h_S$ for $j = 1, \ldots, N$
for $k = 0, \ldots, K-1$ **do**
  parallel for $j = 1, \ldots, N$: compute $\widetilde{h}_{S+j}^{(k)} \leftarrow f_{S+j}(h_{S+j-1}^{(k)})$
  parallel for $j = 1, \ldots, N$: estimate $a_j \leftarrow \mathrm{diag}\Big(J_{S+j}^{(k)}\Big)$ by Hutchinson FD/JVP/VJP probes
  $b_j \leftarrow \widetilde{h}_{S+j}^{(k)} - a_j \odot h_{S+j-1}^{(k)}$ for $j = 1, \ldots, N$
  Solve the affine recurrence $h_{S+j}^{(k+1)} = a_j \odot h_{S+j-1}^{(k+1)} + b_j$ with an associative prefix scan.
end for
return $\mathrm{LMHead}(h_L^{(K)})$

B Analysis Details

B.1 Variance Reduction Derivation

Consider a residual suffix beginning at prefix state h_S, with

$$f_l(h) = h + g_l(h), \qquad l = S+1, \ldots, L. \tag{13}$$

The sequential suffix computes

$$h_L^{\mathrm{seq}} = h_S + \sum_{l=S+1}^{L} g_l(h_{l-1}^{\mathrm{seq}}), \tag{14}$$

where each branch g_l is evaluated at a different accumulated hidden state. In contrast, one-step IDN initialized from h_S computes

$$h_L^{\mathrm{idn}} = h_S + \sum_{l=S+1}^{L} g_l(h_S). \tag{15}$$

Thus the difference is entirely due to evaluation points:

$$h_L^{\mathrm{seq}} - h_L^{\mathrm{idn}} = \sum_{l=S+1}^{L} \left[g_l(h_{l-1}^{\mathrm{seq}}) - g_l(h_S) \right]. \tag{16}$$

First-order expansion around h_S gives

$$g_l(h_{l-1}^{\mathrm{seq}}) - g_l(h_S) \approx J_{g_l}(h_S)\left(h_{l-1}^{\mathrm{seq}} - h_S\right) = J_{g_l}(h_S) \sum_{r=S+1}^{l-1} g_r(h_{r-1}^{\mathrm{seq}}). \tag{17}$$

Sequential execution therefore couples every later layer deviation to all earlier deviations through the branch Jacobians. If the branch contributions have covariance scale σ^2 and $\|J_{g_l}\| \le \rho$ over the suffix, this term contributes a variance scale of order $\rho^2 \sum_l (l-S)\sigma^2$ under a first-order independence approximation. IDN removes this compounding term by evaluating all branches at the same prefix state. The price is bias:

$$\mathrm{Bias}_{\mathrm{idn}} = \sum_{l=S+1}^{L} \left[g_l(h_S) - g_l(h_{l-1}^{\mathrm{seq}}) \right]. \tag{18}$$

SNLP-aware training aims to reduce this bias by encouraging less sensitive branch functions over the suffix trajectory. This first-order argument offers a possible variance-reduction interpretation; the substitution-error diagnostics support reduced sensitivity but do not directly measure this variance.

B.2 Connection to Hyper-Connections and mHC

Hyper-Connections expand the residual stream into M streams and mix them through learned matrices (Zhu et al., 2025). A simplified HC sublayer can be written as

$$x_{l+1} = H_l^{\mathrm{res}} x_l + H_l^{\mathrm{post}} F_l(H_l^{\mathrm{pre}} x_l), \tag{19}$$

where the H matrices act on stream dimension and F_l is the nonlinear branch. For an mHC Transformer block, the attention and MLP wrappers each have their own residual mixing:

$$x'_l = H_{\mathrm{attn},l}^{\mathrm{res}} x_l + H_{\mathrm{attn},l}^{\mathrm{post}} \,\mathrm{Attn}_l\Big(H_{\mathrm{attn},l}^{\mathrm{pre}} x_l\Big), \tag{20}$$

$$x_{l+1} = H_{\mathrm{mlp},l}^{\mathrm{res}} x'_l + H_{\mathrm{mlp},l}^{\mathrm{post}} \,\mathrm{MLP}_l\Big(H_{\mathrm{mlp},l}^{\mathrm{pre}} x'_l\Big). \tag{21}$$

The exact sublayer Jacobians contain both the residual mixing and the nonlinear branch sensitivity:

$$J_{\mathrm{attn},l} = H_{\mathrm{attn},l}^{\mathrm{res}} + H_{\mathrm{attn},l}^{\mathrm{post}} J_{\mathrm{Attn}_l} H_{\mathrm{attn},l}^{\mathrm{pre}}, \tag{22}$$

$$J_{\mathrm{mlp},l} = H_{\mathrm{mlp},l}^{\mathrm{res}} + H_{\mathrm{mlp},l}^{\mathrm{post}} J_{\mathrm{MLP}_l} H_{\mathrm{mlp},l}^{\mathrm{pre}}. \tag{23}$$

If training makes the nonlinear branches locally input-invariant, the branch Jacobian terms become small and the block Jacobian is approximated by

$$J_{\mathrm{block},l} \approx H_{\mathrm{mlp},l}^{\mathrm{res}} H_{\mathrm{attn},l}^{\mathrm{res}}. \tag{24}$$

This is exactly the HCN surrogate used in Eqn. 5. It is available from the architecture and acts only on stream dimension, avoiding full hidden-state Jacobian estimation.

IDN is the degenerate residual case. For $f_l(x) = x + g_l(x)$, the known residual Jacobian is I. IDN uses $A_l = I$ and trains the branch sensitivity J_{g_l} to be small enough that $J_{f_l} = I + J_{g_l} \approx I$. In this sense, IDN and HCN follow the same template: keep the cheap architecture-induced residual transition, and train the nonlinear branch to be compatible with it.

B.3 Fused Cross-Layer Coupling

Layer fusion combines several parallel layers that read the same input h into one wide layer. For a chunk $\mathcal{C}$, the fused attention computes the branch projections for all $l \in \mathcal{C}$ and sums the output projections:

$$a_{\mathcal{C}}(h) = \sum_{l \in \mathcal{C}} O_l \,\mathrm{Attn}_l(Q_l h, K_l h, V_l h). \tag{25}$$

The fused MLP then receives the shared post-attention state

$$u_{\mathcal{C}}(h) = h + a_{\mathcal{C}}(h), \tag{26}$$

and applies the concatenated expansion and down-projection weights, equivalently summing the per-layer MLP branches:

$$m_{\mathcal{C}}(h) = \sum_{l \in \mathcal{C}} P_l \,\phi(W_l \,\mathrm{Norm}(u_{\mathcal{C}}(h))), \tag{27}$$

Attempt	Training target	Observation	Outcome
Vanilla Jacobi	None	$\sigma_{\max} \approx 44$; fixed-point iteration diverges	Failed
Spectral Reg.	Penalize $\mathrm{ReLU}(\widehat{\sigma}_l - 1)$	$\sigma_{\max}$ reduced to ≈ 2.3, but iterations only drop about 20%	Insufficient
IDN/HCN Reg.	Match parallel states to sequential	Removes JVP from correction and makes cheap structured updates effective	Used

Table 5: Summary of early training-side attempts. Spectral regularization improved contractiveness but targeted Jacobian magnitude rather than the practical bottleneck: computing or estimating Jacobian corrections during inference.

Model	Regularization	Steps	N	λ	Stride	Detach
Nanochat-3B standard	None	9600	–	–	–	–
	IDN	9600	8	0.5	0	✗
Nanochat-0.5B standard	None	4800	–	–	–	–
	IDN	4800	24	0.0625	3	✗
Nanochat-0.5B w/o x0ve	None	4800	–	–	–	–
	IDN	4800	24	0.5	6	✗
Nanochat-0.5B-mHC	None	4800	–	–	–	–
	HCN	4800	24	0.5	0	✓

Table 6: Training and SNLP-aware regularization settings for models used in the main results. Detach indicates whether the sequential target in Eqn. 10 is detached.

where ϕ is the MLP nonlinearity. The fused chunk output is

$$F_{\mathcal{C}}^{\text{fused}}(h) = u_{\mathcal{C}}(h) + m_{\mathcal{C}}(h). \tag{28}$$

This is not identical to independently evaluating and summing each layer branch, because every MLP branch sees the aggregate attention state $h + \sum_{r \in \mathcal{C}} a_r(h)$ rather than only its own $h + a_l(h)$. The cross-layer coupling term is

$$\sum_{l \in \mathcal{C}} P_l \left[\phi\left(W_l \,\mathrm{Norm}\left(h + \sum_{r \in \mathcal{C}} a_r(h) \right) \right) - \phi(W_l \,\mathrm{Norm}(h + a_l(h))) \right]. \tag{29}$$

This term captures the change induced by evaluating every MLP branch on the aggregate post-attention state rather than on its own per-layer post-attention state. It is generally nonzero: layer normalization, nonlinear MLP activations, and non-canceling projections can all make a branch react to attention evidence produced by other layers in the same chunk. This implicit cross-layer coupling connects fused SNLP to the broader observation that architectures can expose useful depth mixing, as in HC/mHC and AttnRes (Zhu et al., 2025; Xie et al., 2025; Team et al., 2026), while also explaining why overly aggressive fusion can change model behavior.

C Training Configuration and Ablations

Table 5 summarizes an early training-side ablation that motivated SNLP-aware regularization. Vanilla Jacobi, which initializes every layer state from the prefix state and repeatedly applies $h_l^{(k+1)} = f_l(h_{l-1}^{(k)})$, diverged on trained models: measured layer Jacobian norms were far from contractive. Spectral regularization reduced this norm substantially, but it only modestly reduced the number of Newton iterations and did not address the dominant cost of Jacobian estimation. This motivated the IDN/HCN direction: instead of making exact Newton cheaper, train the model so a cheap structured correction is useful.

Table 6 lists the training settings used for the trained-from-scratch models in Table 1. Table 7 shows that the best loss configuration depends on architecture and scale: the 3B model prefers a smaller parallel suffix, the 0.5B standard model is sensitive to both λ and stride, and the no-x0/VE model works best with stride 6. Detaching the matching target hurts IDN PPL, while HCN on the mHC model uses a detached target by default.

Model	Regularization	Steps	N	λ	Stride	Detach	PPL	ΔPPL
Nanochat-3B standard [No Reg. PPL: 10.10]	IDN	9600	8	0.5	0	✗	10.07	-0.3%
	IDN	9600	24	0.1	0	✗	10.12	+0.2%
	IDN	9600	24	0.1	3	✗	10.59	+4.9%
	IDN	9600	24	0.1	6	✗	10.36	+2.6%
	IDN	9600	24	0.5	0	✗	11.41	+13.0%
	DiagN	9600	8	0.05	3	✓	10.23	+1.3%
	DiagN	9600	8	0.1	3	✓	10.26	+1.6%
	DiagN	9600	24	0.5	3	✓	10.31	+2.1%
	DiagN	9600	24	0.1	3	✓	10.08	-0.2%
	DiagN	9600	8	0.5	3	✓	10.43	+3.3%
Nanochat-0.5B standard [No Reg. PPL: 15.21]	IDN	4800	24	0.5	0	✗	15.23	+0.1%
	IDN	4800	24	0.1	0	✗	15.34	+0.9%
	IDN	4800	24	0.0625	3	✗	15.36	+1.0%
	IDN	4800	24	0.1	3	✗	15.43	+1.5%
	IDN	4800	24	0.5	0	✓	16.46	+8.3%
	IDN	4800	24	0.5	3	✗	16.81	+10.5%
	IDN	4800	24	0.5	2	✗	21.36	+40.5%
	IDN	4800	24	0.5	3	✓	NaN	–
	IDN	2040	24	0.5	0	✗	17.54	+15.3%
	DiagN	4800	24	0.1	3	✓	15.18	-0.2%
	DiagN	4800	24	0.5	3	✓	17.13	+12.6%
	DiagN	2040	24	0.1	3	✓	17.08	+12.3%
	DiagN	2040	24	0.5	3	✓	26.70	+75.5%
Nanochat-0.5B w/o x0ve [No Reg. PPL: 17.65]	IDN	4800	24	0.5	6	✗	17.57	-0.5%
	IDN	4800	24	0.5	0	✗	17.63	-0.1%
	IDN	4800	24	0.5	3	✗	20.33	+15.2%
Nanochat-0.5B-mHC [No Reg. PPL: 15.16]	HCN	4800	24	0.5	0	✓	15.52	+2.4%
	HCN	4800	24	0.5	0	✗	15.91	+4.9%
	HCN	4800	24	0.5	3	✓	21.67	+43.0%

Table 7: Ablation of SNLP-aware regularization loss settings. Detach indicates whether the sequential target is detached. PPL is evaluated at sequence length 2048. ΔPPL is relative to the corresponding No Reg. baseline shown under each model group. Selected main-result checkpoints are highlighted.

D Downstream Evaluation and Self-Speculative Decoding

D.1 Downstream Task Evaluation

For downstream evaluation, we follow Nanochat (Karpathy, 2025) to SFT on SmolTalk (Allal et al., 2025) and evaluate multiple-choice accuracy on ARC (Clark et al., 2018) and MMLU (Hendrycks et al., 2021).

Table 8 separates preservation of IDN compatibility from absolute downstream accuracy. After sequential SFT, switching only inference to IDN reduces 0.5B base-model accuracy from 38.36% to 30.02% (8.34 points), versus 36.15% to 35.78% (0.37 points) for the regularized model. At 3B, the corresponding drops are 57.85% to 54.60% (3.25 points) and 57.41% to 57.03% (0.38 points). Thus IDN-regularized pretraining better preserves compatibility after standard SFT, but absolute gaps remain: 57.03% versus the 57.85% base sequential pipeline at 3B, and 36.23% versus 38.36% for 0.5B IDN-forward SFT with IDN inference.

At fixed `IDN-12xF1-K2` inference, SNLP-forward SFT improves 0.5B accuracy from 30.02% to 34.38% without regularization and from 35.78% to 36.23% with regularization. SNLP-forward SFT on the 3B regularized checkpoint uses `IDN-12xF1-K1`; the same $N = 8$ parallelism as pretraining gave too small an initial loss to provide a useful training signal in our runs. Its different parallel depth prevents a direct comparison of SFT methods with the `IDN-8xF1-K1` inference row.

Model	SFT Method	Inference Method	ARC-Easy	ARC-Challenge	MMLU	Avg
Nanochat-3B standard						
No Reg. [10.10]	Sequential	Sequential	74.91%	58.86%	39.78%	**57.85%**
	Sequential	IDN-8xF1-K1	70.70%	54.18%	38.93%	54.60%
	IDN-8xF1-K1	Sequential	72.46%	51.84%	39.71%	54.67%
	IDN-8xF1-K1	IDN-8xF1-K1	73.33%	54.85%	39.71%	55.96%
IDN Reg. [10.07]	Sequential	Sequential	74.74%	57.19%	40.30%	**57.41%**
	Sequential	IDN-8xF1-K1	74.39%	56.86%	39.84%	57.03%
	IDN-12xF1-K1	Sequential	70.88%	55.52%	37.95%	54.78%
	IDN-12xF1-K1	IDN-12xF1-K1	70.88%	55.85%	38.08%	54.94%
Nanochat-0.5B standard						
No Reg. [15.21]	Sequential	Sequential	44.91%	39.13%	31.03%	**38.36%**
	Sequential	IDN-12xF1-K2	31.40%	30.43%	28.22%	30.02%
	IDN-12xF1-K2	Sequential	38.42%	31.44%	31.74%	33.87%
	IDN-12xF1-K2	IDN-12xF1-K2	39.30%	32.11%	31.74%	34.38%
IDN Reg. [15.36]	Sequential	Sequential	37.72%	38.80%	31.94%	36.15%
	Sequential	IDN-12xF1-K2	38.25%	37.79%	31.29%	35.78%
	IDN-12xF1-K2	Sequential	43.16%	35.79%	30.50%	**36.48%**
	IDN-12xF1-K2	IDN-12xF1-K2	42.28%	35.79%	30.63%	36.23%

Table 8: Downstream multiple-choice accuracy after SFT with either sequential or SNLP forward. Each checkpoint is evaluated with both sequential and IDN inference.

D.2 Self-Speculative Decoding

We also use SNLP as a drafter inside self-speculative decoding, with the same model's sequential forward as the verifier (Leviathan et al., 2023; Chen et al., 2023). The verifier guarantees identical output to sequential greedy decoding, while SNLP changes draft cost and acceptance. For block size B, drafter forward speedup s, and per-draft greedy acceptance rate α, we estimate ideal speedup as $((1-\alpha^{B+1})/(1-\alpha))/(B/s+1)$. Table 9 evaluates 128 validation prompts with prefill length 16 and 1024 greedy generated tokens. We report greedy results because sampling substantially reduces acceptance rates in our runs. The best configuration reaches $1.67\times$ estimated speedup: a fast `12xF2` drafter compensates for lower acceptance. The optimal block size is usually $B = 4$, while very high-acceptance mHC and higher-K IDN configs prefer $B = 8$. Increasing Newton iterations improves α but slows the drafter, so the net speedup can decrease. HCN maintains especially high greedy acceptance despite nonzero PPL gap, suggesting that it preserves token ranking well.

Config	s	B	α	Est. speedup
Nanochat-0.5B standard, IDN Reg.				
`24xF1-K2-h0`	1.86×	2	0.969	1.401×
		4	0.963	1.474×
		8	0.952	1.406×
`24xF1-K4-h0`	1.30×	2	0.999	1.181×
		4	0.997	1.219×
		8	0.995	1.233×
Nanochat-0.5B w/o x0ve, IDN Reg.				
`12xF2-K1-h0`	2.63×	2	0.925	1.579×
		4	0.914	1.670×
		8	0.889	1.456×
`12xF2-K2-h0`	2.15×	2	0.965	1.500×
		4	0.958	1.607×
		8	0.942	1.519×
Nanochat-0.5B-mHC, HCN Reg.				
`16xF1-K1-h0`	1.63×	2	0.998	1.344×
		4	0.997	1.439×
		8	0.995	1.493×
`16xF1-K2-h0`	1.44×	2	0.999	1.255×
		4	0.999	1.321×
		8	0.997	1.357×

Table 9: Self-speculative decoding with sequential verification. Forward speedup s and acceptance α are measured; B is draft block size. Speedup is estimated from displayed inputs using the stated model, not measured end-to-end.

D.3 Comparison with Multi-Token Prediction

Multi-token prediction (MTP) can provide an auxiliary training objective or a speculative drafter through additional prediction heads (Gloeckle et al., 2024). Standard next-token PPL alone does not characterize the latter use, so we compare MTP and IDN through verified greedy decoding. Each drafter is verified by its own model's sequential forward, preserving that model's greedy output; this does not imply identical outputs across differently trained models.

We train matched Nanochat-0.5B models for 4,800 steps with MTP-1, IDN regularization, or both. Evaluation uses 128 samples, a 16-token prefill, and 256 generated tokens without KV caching. All configurations use draft block size $B = 2$; IDN parallelizes the final 24 of 32 layers from `h0`. This protocol differs from the 1024-token sweep in Table 9.

Table 10 reports measured per-token acceptance α, its standard deviation, measured output tokens per verification round (Tok/Rnd), and the wall time of the complete evaluation. Estimated speedup uses

$$\widehat{S} = \frac{\sum_{i=0}^{B} \alpha^i}{C_{\text{round}}}, \qquad C_{\text{round}} = \begin{cases} B/s + 1, & \text{IDN drafter,} \\ c_{\text{MTP}} + 1, & \text{MTP-based drafter.} \end{cases}$$

Here s is verifier time divided by IDN draft time, and c_{MTP} is the MTP-based drafter's time divided by verifier time, including IDN when used. Both costs are measured at sequence length 128. The numerator is an idealized expectation from α, distinct from measured Tok/Rnd; estimated speedups are not ratios of complete-evaluation wall times.

Training	Drafter	K	α	Std.	Tok/Rnd	Est. speedup	Wall time (s)
MTP-1	MTP	–	0.987	0.024	2.98	1.494×	318.9
IDN + MTP-1	IDN	2	0.986	0.024	2.98	1.435×	345.0
IDN + MTP-1	IDN	4	0.996	0.009	3.00	1.180×	456.0
IDN	IDN	2	0.978	0.083	2.97	1.419×	343.2
IDN	IDN	4	0.996	0.017	3.00	1.170×	449.2
IDN + MTP-1	MTP+IDN	2	0.978	0.031	2.97	1.893×	244.0
IDN + MTP-1	MTP+IDN	4	0.986	0.029	2.99	1.658×	272.4

Table 10: MTP, IDN, and combined drafters under verified greedy decoding. Acceptance, Tok/Rnd, and complete-evaluation wall time are measured; speedup is estimated from acceptance and separately timed draft costs.

Separately trained MTP and IDN models reach best estimated speedups of $1.494\times$ and $1.419\times$, respectively. An IDN-only drafter on the jointly IDN+MTP-trained checkpoint reaches $1.435\times$. They target different axes: MTP predicts future tokens, while IDN shortens the layerwise critical path within a forward. The combined drafter reaches $1.893\times$ estimated speedup and the lowest measured evaluation time, 244.0 seconds. Increasing IDN from $K = 2$ to $K = 4$ improves acceptance but increases runtime. These results support complementarity in this research implementation, rather than establishing performance in an optimized serving system.

E Additional Inference Ablations

Checkpoint note. The correction-ordering and correction-propagation ablations in this appendix use the more parallelizable 0.5B IDN checkpoint from Table 7 ($N = 24$, $\lambda = 0.5$, stride 3, no detach; PPL 16.81).

E.1 Interpreting AR Match Rate

Table 11 gives a rough conversion between the observed greedy autoregressive match rate and an effective local per-token agreement rate. For AR Match, each sample is evaluated over $T = 32$ generated tokens. The reported AR match rate α is token-level agreement against the sequential baseline over N samples $\alpha = \frac{\#\text{matched generated tokens}}{NT}$. Because generation is autoregressive, a single mismatch changes the prefix for all later positions. Under a simple absorbing-divergence model, let β

Observed AR match α	Implied conditional match $\widehat{\beta}$
0.95	0.9969
0.90	0.9935
0.80	0.9861
0.70	0.9773
0.60	0.9667
0.50	0.9533
0.40	0.9356
0.30	0.9098

Table 11: Conversion from observed token-level AR match rate to effective conditional per-token agreement for $T = 32$ generated tokens under an absorbing-divergence model.

be the probability of matching the next token conditioned on all previous generated tokens matching. Then token i matches with probability β^i, so

$$\alpha = \tfrac{1}{T}\sum_{i=1}^{T}\beta^i = \tfrac{\beta(1-\beta^T)}{T(1-\beta)}.$$

Thus an apparently modest AR match can still imply high local agreement before prefix divergence; for $T = 32$, $\alpha = 0.30$ corresponds to $\widehat{\beta} \approx 0.910$.

E.2 Coupling SNLP with Jacobi Decoding

SNLP parallelizes over the layer axis, while Jacobi decoding (JD) parallelizes over future token positions by iteratively refining a block of draft tokens (Santilli et al., 2023). Speculative decoding and speculative Jacobi decoding (SJD) further combine draft proposals with verification (Leviathan et al., 2023; Chen et al., 2023; Teng et al., 2025). A natural extension is to couple the two axes and solve over a layer-token lattice

$$h_{\ell,t}, \qquad \ell = 0, \ldots, L, \quad t = 1, \ldots, T.$$

In principle, one could update hidden states and token guesses jointly,

$$h_{\ell,t}^{(r+1)} = \mathcal{F}_{\ell,t}\Big(h^{(r)}, x^{(r)}\Big), \qquad x_t^{(r+1)} = \mathrm{JDUpdate}\Big(\mathrm{LMHead}(h_{L,t}^{(r+1)})\Big),$$

rather than using a nested loop,

$$x^{(r+1)} \leftarrow \mathrm{JDUpdate}\Big(\mathrm{SNLPForward}(x^{(r)}; K)\Big).$$

This view is appealing because a successful 2D solver could avoid paying a full inner SNLP solve for every token-level Jacobi iteration. In practice, the main difficulty is initialization. JD changes the draft tokens between iterations; reinitializing all layer states from the new prefix state h_0 reduces to the nested baseline, while reusing hidden states from the previous JD iteration carries features computed for old draft tokens.

Table 12 evaluates this design on a 0.5B IDN stride-0 checkpoint. The naive variants run a fresh SNLP forward inside each JD iteration. The h0-JD variants recompute the first parallel input but warm-start deeper parallel states from the previous JD iteration. Naive $K = 1$ already obtains 100% token match across all tested configurations, leaving no inner-iteration gap for coupling to close. Increasing to $K = 2$ adds cost without improving match. By contrast, h0-JD degrades match to roughly 21–37%, because stale hidden states encode features for old draft tokens and are propagated by the IDN correction. SJD-style variable-length acceptance makes direct hidden-state reuse even less straightforward, so the tested SJD path effectively reduces to naive composition. We therefore leave useful 2D layer-token coupling to future work; it likely requires a better transport or reinitialization rule when draft tokens change.

Config	8xF1, $N = 8$				12xF1, $N = 12$				4xF3, $N = 12$			
	Match	Accept	JD iters	Fwd passes	Match	Accept	JD iters	Fwd passes	Match	Accept	JD iters	Fwd passes
naive $K = 1$	**100.0%**	**1.04**	**1.96**	**30.9**	**100.0%**	**1.13**	**1.91**	**28.8**	**100.0%**	**1.05**	**1.96**	**30.4**
naive $K = 2$	100.0%	1.02	1.98	31.5	100.0%	1.04	1.95	30.8	100.0%	1.06	1.93	30.2
h0-JD $K = 1$	25.0%	0.85	2.37	37.9	21.2%	0.70	2.88	45.9	22.8%	1.10	2.16	31.0
h0-JD $K = 2$	37.2%	1.00	2.07	31.9	27.8%	0.94	2.17	34.2	36.5%	1.03	2.01	31.1

Table 12: Coupling JD with SNLP on a 0.5B IDN stride-0 checkpoint, evaluated on 8 prompts with 32 generated tokens and lookahead window 5. Naive runs reinitialize SNLP hidden states in each JD iteration. `h0-JD` recomputes the first parallel input but reuses deeper parallel states from the previous JD iteration.

ELK	Init	1xF12			2xF6			4xF3			12xF1		
		h0	fwd	preheat	h0	fwd	preheat	h0	fwd	preheat	h0	fwd	preheat
0	PPL	94.04	94.04	94.04	102.8	102.5	169.1	105.8	94.44	281.0	144.5	93.83	1218
0.1	PPL	94.04	94.04	94.04	97.94	93.90	124.7	91.75	82.43	124.2	586.5	92.87	81293

Table 13: ELK tempering and preheat initialization ablation for $N = 12$, $K = 1$, evaluated at sequence length 128 with sequential PPL 59.07. Configs follow the main chunk notation: `1xF12`, `2xF6`, `4xF3`, and `12xF1`. The `fwd` column corresponds to one-shot batched-forward initialization.

E.3 ELK Tempering and Preheat Initialization

We also explored two auxiliary inference knobs that are not included in the main configuration search. The first is an ELK-style tempering of the Newton correction (Gonzalez et al., 2024), which scales the correction term without changing the number of block forwards and therefore should not affect speed. The second is preheat initialization: offline calibration fits a low-rank affine predictor for each layer output,

$$\widehat{h}_l(x_0) = (x_0 V_l) U_l^\top + b_l,$$

where the basis V_l is obtained from a truncated SVD of calibration embeddings and U_l, b_l are fit by linear regression to sequential hidden states. At inference time, $\widehat{h}_l(x_0)$ initializes the parallel suffix. In the table below, preheat was calibrated on random tokens; later validation-set calibration did not consistently improve results.

Table 13 reports an auxiliary sequence-length-128 sweep and shows that ELK tempering can improve the PPL–iteration tradeoff for some configurations, but the effect is not uniform. Preheat is also inconsistent: it can be close to `h0`, but it can also be much worse. To reduce search space, our main experiments do not tune ELK or preheat; a more systematic search may expose different points on the speed-quality frontier.

E.4 Off-the-Shelf Model Results

Table 14 summarizes selected post-hoc SNLP configurations for off-the-shelf models from the L1 timing sweep. We report fast configurations under relaxed PPL thresholds and the lowest-PPL configuration. Matching sequential perplexity requires multiple Newton iterations and does not produce speedup, supporting the need for SNLP-aware training.

E.5 Diagonal-Jacobian Correction Results

Table 16 benchmarks the local cost of Jacobian-vector primitives used by the diagonal-correction sweep in Table 15. Finite-difference (FD) estimates Jv by evaluating $f(x + \epsilon v) - f(x)$, exact forward-mode JVP returns both $f(x)$ and Jv in one call, and VJP uses reverse-mode autodiff to compute $J^\top v$. FD is efficient in our solver because $f(x)$ is already computed by the parallel block forward, so the finite difference only adds the extra $f(x + \epsilon v)$ evaluation, similar to the cost of forward initialization. However, FD often diverges in our sweep, plausibly because the correction stacks multiple approximations: replacing the full Jacobian by its diagonal, estimating that diagonal with a Hutchinson-style probe, and then using a noisy finite difference. JVP and VJP are more stable

Model	Seq PPL	Config	K	PPL	Speedup
Qwen2.5-0.5B	14.36	`8xF1-fwd`	4	15.55 (+8.2%)	0.74×
		`8xF1-fwd`	8	14.36 (+0.0%)	0.64×
TinyLlama-1.1B	8.03	`8xF1-fwd`	4	8.84 (+10.1%)	0.75×
		`12xF1-fwd`	8	8.17 (+1.7%)	0.66×
		`8xF1-fwd`	8	8.03 (+0.0%)	0.63×

Table 14: Selected post-hoc SNLP configurations on off-the-shelf models. Config names follow the chunk notation used in Table 1; `8xF1-fwd` denotes 8 parallel single-layer chunks initialized from a one-shot parallel forward.

Model	Config	K	PPL	ΔPPL	Speedup	Top-1	LogitSim	EmbSim	AR Match
Nanochat-3B standard									
No Reg.	`8xF1-h0`	4	10.25	+1.5%	0.44×	92.2%	0.997	0.985	80.2%
[10.10]	`8xF1-fwd`	4	10.20	+1.0%	0.42×	96.1%	0.999	0.990	88.9%
IDN Reg.	`8xF1-h0`	1	10.50	+4.3%	0.71×	85.8%	0.985	0.968	63.3%
[10.07 (-0.3%)]	`8xF1-h0`	4	10.10	+0.3%	0.44×	98.1%	1.000	0.993	93.4%
Nanochat-0.5B standard									
No Reg.	`8xF1-fwd`	2	16.65	+9.5%	0.58×	82.4%	0.992	0.981	71.2%
[15.21]	`8xF1-fwd`	4	15.45	+1.6%	0.44×	98.3%	1.000	0.998	97.3%
IDN Reg.	`8xF1-h0`	1	16.30	+6.1%	0.72×	86.4%	0.991	0.984	79.9%
[15.36 (+1.0%)]	`8xF1-h0`	2	15.53	+1.1%	0.60×	96.1%	0.999	0.994	92.2%
Nanochat-0.5B w/o x0ve									
No Reg.	`8xF1-h0`	2	19.35	+9.6%	0.62×	84.9%	0.997	0.985	78.3%
[17.65]	`8xF1-h0`	4	17.76	+0.6%	0.48×	97.7%	1.000	0.999	97.9%
IDN Reg.	`20xF1-h0`	2	18.80	+7.0%	0.88×	77.2%	0.996	0.960	60.9%
[17.57 (-0.5%)]	`8xF1-h0`	2	17.81	+1.4%	0.62×	90.9%	1.000	0.982	79.7%
Nanochat-0.5B-mHC									
No Reg.	`8xF1-h0`	4	16.03	+5.7%	0.46×	92.2%	0.996	0.992	87.6%
[15.16]	`8xF1-fwd`	4	15.34	+1.2%	0.45×	96.9%	0.999	0.997	94.8%
HCN Reg.	`8xF1-h0`	8	15.87	+2.2%	0.31×	92.6%	0.998	0.988	86.0%
[15.52 (+2.4%)]									

Table 15: Main diagonal-Jacobian Newton correction results using 2048-token PPL evaluation. Brackets under model names give sequential PPL and, when shown, the change from the corresponding No Reg. baseline. Configs use the same chunk notation as Table 1; all rows use DiagN correction.

but expensive: Table 16 shows roughly 3× overhead for VJP and 6–8× for JVP over a plain forward. Exact JVP also lacks fused-SDPA support, so the diagonal experiments use FD or VJP estimators.

In a short-context mHC diagnostic, no diagonal Newton configuration reaches the 10% fast threshold relative to sequential PPL 88.67: the best diagonal result reaches 104.2 PPL, or +17.5% above sequential. With the x0+VE mHC models in Table 15, diagonal correction can match PPL but remains much slower. Although VJP computes $J^\top v$ rather than Jv, this alone does not imply diagonal-estimator bias: $\mathrm{diag}(J^\top) = \mathrm{diag}(J)$, and both Hutchinson-style estimators are unbiased with isotropic probes in exact arithmetic. Finite-probe variance and the diagonal approximation may affect solver behavior; these experiments do not isolate the cause of instability.

E.6 Pretrained TinyLlama Fine-Tuning

We briefly finetune TinyLlama-1.1B-Chat-v1.0 (Zhang et al., 2024) with IDN regularization to test whether layer-parallel compatibility can be retrofitted onto an off-the-shelf pretrained model. TinyLlama is a standard LLaMA-style model with 22 layers, hidden dimension 2048, 32 attention

Model	T	Forward (ms)	FD (ms)	JVP (ms)	VJP (ms)
Nanochat-3B standard	1	0.458	0.816 (1.78×)	3.274 (7.14×)	1.346 (2.94×)
	16	0.459	0.959 (2.09×)	3.356 (7.31×)	1.369 (2.98×)
	32	0.458	0.926 (2.02×)	3.441 (7.51×)	1.379 (3.01×)
	64	0.463	0.952 (2.05×)	3.421 (7.38×)	1.355 (2.92×)
	128	0.454	1.006 (2.22×)	3.380 (7.44×)	1.345 (2.96×)
Nanochat-0.5B standard	1	0.418	0.784 (1.88×)	3.235 (7.74×)	1.330 (3.18×)
	16	0.420	0.833 (1.98×)	3.299 (7.85×)	1.292 (3.08×)
	32	0.411	0.845 (2.06×)	3.378 (8.23×)	1.339 (3.26×)
	64	0.424	0.872 (2.05×)	3.321 (7.83×)	1.309 (3.08×)
	128	0.421	0.832 (1.98×)	3.397 (8.07×)	1.334 (3.17×)
Qwen2.5-0.5B	1	0.522	1.059 (2.03×)	3.014 (5.77×)	1.753 (3.35×)
	16	0.545	1.091 (2.00×)	3.132 (5.75×)	1.855 (3.40×)
	32	0.545	1.090 (2.00×)	3.179 (5.84×)	1.875 (3.44×)
	64	0.542	1.071 (1.97×)	3.127 (5.77×)	1.843 (3.40×)
	128	0.537	1.080 (2.01×)	3.162 (5.89×)	1.852 (3.45×)

Table 16: Single-layer operation benchmark. Nanochat timings use layer 24, and Qwen2.5 timings use a standalone middle layer. Times are milliseconds over 100 warmup and 500 timed runs; parentheses report cost relative to a plain forward.

Run	IDN weight	Target N	LR	Steps	Final PPL
baseline	0	–	2×10^{-5}	2000	17.82
idn05	0.5	4,8,12	2×10^{-5}	2000	17.91
idn2_npar48	2.0	4,8	5×10^{-5}	5000	19.29
idn5_npar4	5.0	4	1×10^{-4}	5000	26.03
idn10_npar4	10.0	4	1×10^{-4}	5000	26.04
idn5_npar4812	5.0	4,8,12	5×10^{-5}	5000	19.53
idn1_long	1.0	4,8	1×10^{-5}	4000*	≈ 17.8
idn5_long_npar4	5.0	4	1×10^{-5}	4000*	≈ 17.8

Table 17: TinyLlama IDN finetuning grid. Final PPL uses sequence length 128 for quick early testing. Runs marked with * crashed before the planned 10000 steps; the checkpoint near 2000 steps was still usable for evaluation.

heads, and 4 KV heads. We finetune on ClimbMix (Diao et al., 2025) using AdamW with cosine learning-rate decay; Table 17 summarizes the small grid.

These preliminary runs support the co-design interpretation: simple post-hoc finetuning is hard to use to change a pretrained model's layer-parallel behavior when the model was not pretrained with SNLP regularization. We leave stronger pretrained-model adaptation to future investigation.

E.7 Correction Ordering

Tables 18 to 20 report the correction-ordering ablations used to construct the summary in Table 3. AR-ness is the average of local- and global-AR-ness scores as defined in DiffuCoder (Gong et al., 2025). At $K = 1$, several non-forward orders remain usable; after repeated correction at $K = 4$, forward ordering consistently gives the best stable result, while most non-forward orders degrade or diverge.

E.8 Correction Propagation

Table 21 compares subset ablations with and without Newton correction. The identical columns in the no-correction block show that information advances only one layer per iteration, while the correction block shows that all active suffix layers can influence the output immediately.

		IDN Reg.			No Reg.		
Order	AR-ness	Seq-Perm	$K=1$	$K=4$	Seq-Perm	$K=1$	$K=4$
forward	1.000	**16.8**	**17.2**	**16.9**	**15.2**	1585.9	**15.4**
shuffle_15	0.384	16.8	24.4	281.6	98.1	32.1	496161.9
shuffle_0	0.339	16.8	19.5	46.4	2391.2	210.3	13103.0
shuffle_14	0.321	16.9	29.3	28.3	579.7	32.8	444232.8
shuffle_5	0.259	16.8	18.2	19.1	249.2	296.0	div.
shuffle_10	0.259	16.8	17.6	19.4	32.8	667.8	div.
shuffle_13	0.205	16.8	18.5	22.1	530.0	12736.3	div.
shuffle_9	0.196	16.8	20.2	div.	97.7	41.4	240814.7
shuffle_6	0.188	16.8	27.2	div.	39.5	28.1	div.
shuffle_11	0.188	16.9	29.3	32.0	654.4	32.8	768834.1
shuffle_12	0.188	16.8	23.7	44.2	40.7	28.7	139401.7
shuffle_2	0.134	16.8	18.2	41.5	91.4	29796.1	92160.6
shuffle_1	0.125	16.9	41.3	54.1	1028.0	32.9	241407.0
shuffle_3	0.125	16.9	22.9	32.3	48.1	**27.2**	div.
shuffle_4	0.125	16.8	28.5	div.	359.9	34.3	9585.0
shuffle_7	0.125	16.9	36.3	22.7	6281.4	33.9	185013.0
shuffle_8	0.125	16.9	22.3	50.7	156.8	30.5	div.
reversed	0.062	16.9	37.4	41.7	24622.8	37.3	div.
no correction	–	–	37.4	19.7	–	37.3	20.0

Table 18: Correction ordering ablation for $N = 8$ on the IDN Reg. checkpoint and No Reg. baseline, evaluated at sequence length 2048.

		IDN Reg.			No Reg.		
Order	AR-ness	Seq-Perm	$K=1$	$K=4$	Seq-Perm	$K=1$	$K=4$
forward	1.000	**16.8**	17.9	**17.0**	**15.2**	283.7	**18.2**
shuffle_7	0.390	17.0	31.7	37.4	15111.5	177.7	610793.2
shuffle_10	0.341	17.0	117.4	1210.0	90291.2	184.9	div.
shuffle_13	0.299	16.9	17.9	141.2	94.5	283.6	div.
shuffle_0	0.261	16.9	32.1	div.	11287.2	567.3	div.
shuffle_6	0.254	16.9	20.8	div.	135.9	95.5	div.
shuffle_9	0.212	16.9	23.7	div.	1259.9	81.3	div.
shuffle_5	0.170	16.9	**17.8**	23.2	82.0	283.6	div.
shuffle_11	0.167	16.9	28.4	441.4	10006.0	**71.5**	div.
shuffle_3	0.125	17.1	67.2	25.9	82494.6	219.7	247566.1
shuffle_4	0.125	16.9	67.2	27.6	1452.9	219.7	div.
shuffle_12	0.125	17.0	52.0	div.	3809.4	1286.6	div.
shuffle_1	0.087	16.9	19.2	98879.6	491.6	1105.8	div.
shuffle_2	0.083	17.0	28.7	div.	50156.7	346.7	div.
shuffle_14	0.083	17.0	35.4	div.	15778.7	72.6	div.
shuffle_8	0.042	17.0	21.9	div.	2751.0	1182.6	div.
shuffle_15	0.042	16.9	27.6	808.6	16138.4	73.8	div.
reversed	0.042	17.2	67.2	82.5	843995.5	219.7	165382.1
no correction	–	–	67.2	30.0	–	219.7	46.8

Table 19: Correction ordering ablation for $N = 12$ on the IDN Reg. checkpoint and No Reg. baseline, evaluated at sequence length 2048.

F Efficiency Regime and Additional Measurements

SNLP targets latency-sensitive inference with spare parallel capacity. It shortens the sequential critical path across layers but need not reduce total computation: $K > 1$ evaluates the parallel suffix repeatedly, and layer-parallel execution increases activation memory. Its benefit therefore depends

		IDN Reg.			No Reg.		
Order	AR-ness	Seq-Perm	$K=1$	$K=4$	Seq-Perm	$K=1$	$K=4$
forward	1.000	**16.8**	**18.0**	**17.0**	**15.2**	186.4	963.3
shuffle_7	0.254	25.6	1158.1	2958.4	64072.0	1179.2	div.
shuffle_15	0.225	18.7	76.5	div.	779110.9	1044.2	div.
shuffle_10	0.190	47.2	115.5	div.	9214.1	173.0	div.
shuffle_13	0.190	24.2	53.0	4823.7	206.1	**126.7**	div.
shuffle_2	0.156	19.4	86.9	75013.0	844459.1	802.1	div.
shuffle_14	0.131	319.6	div.	9509.3	68188.1	198.2	div.
shuffle_0	0.129	11585.0	div.	div.	73512.0	310.8	div.
shuffle_3	0.127	26.5	1625.1	div.	60895.9	1257.8	div.
shuffle_4	0.127	21.0	38.0	div.	22761.7	400.9	div.
shuffle_6	0.127	18.3	div.	659.0	399655.1	711.5	div.
shuffle_9	0.125	35.6	div.	441.5	262632.2	1847.5	div.
shuffle_12	0.125	17.0	60.2	div.	71510.7	853.7	div.
shuffle_5	0.094	21.9	30.0	24986.9	56120.8	233.9	div.
shuffle_8	0.094	22.0	div.	320.6	47150.1	2951.5	div.
shuffle_1	0.062	61.0	div.	229855.1	20786.3	625.6	div.
shuffle_11	0.062	33.3	978.1	div.	12269.6	223.4	div.
reversed	0.031	316.4	div.	div.	div.	div.	43424.8
no correction	–	–	div.	div.	–	5382.8	**328.6**

Table 20: Correction ordering ablation for $N = 16$ on the IDN Reg. checkpoint and No Reg. baseline, evaluated at sequence length 2048.

Correction	K	last 1	last 2	last 3	last 4	last 5	last 6	last 7	last 8
w/ Corr.	1	38.70	25.64	22.32	20.57	19.44	18.71	17.89	17.19
	2	38.56	25.71	22.23	20.26	19.11	18.37	17.45	16.91
	3	38.59	25.71	22.22	20.27	19.12	18.38	17.46	16.91
	4	38.59	25.71	22.22	20.27	19.12	18.38	17.46	16.91
	5	38.59	25.71	22.22	20.27	19.12	18.38	17.46	16.91
	6	38.59	25.71	22.22	20.27	19.12	18.38	17.46	16.91
	7	38.59	25.71	22.22	20.27	19.12	18.38	17.46	16.91
	8	38.59	25.71	22.22	20.27	19.12	18.38	17.46	16.91
w/o Corr.	1	37.41	37.41	37.41	37.41	37.41	37.41	37.41	37.41
	2	32.07	22.97	22.97	22.97	22.97	22.97	22.97	22.97
	3	37.58	25.21	21.90	21.90	21.90	21.90	21.90	21.90
	4	32.98	23.40	21.01	19.73	19.73	19.73	19.73	19.73
	5	37.74	25.27	21.95	19.86	18.88	18.88	18.88	18.88
	6	37.83	25.31	21.97	19.87	18.89	18.17	18.17	18.17
	7	49.44	28.55	25.01	21.06	19.69	18.75	17.72	17.72
	8	38.59	25.71	22.22	20.27	19.12	18.38	17.46	16.91

Table 21: Subset ablation with and without Newton correction for the 0.5B IDN Reg. model at sequence length 2048. Columns activate progressively more suffix layers ending at layer 31. With correction, all active layers affect PPL at $K = 1$. Without correction, influence propagates one layer per iteration: at $K = 1$ all columns are identical, at $K = 2$ columns last 2–last 8 are identical, and so on.

on hardware utilization and the fraction of depth that training makes parallelizable. The following profiling and FHE-oriented simulation examine these costs in two different execution regimes.

F.1 Component Profiling and Activation Memory

We profile Nanochat-0.5B and Nanochat-3B standard models on an H100 at batch size 1 and sequence length 128 in BF16, with 50 warmup and 200 measured runs. Tables 22 and 23 report component latency, PPL, and activation memory. Component timing synchronizes individual operations; `bench()` independently measures a complete forward without per-component synchronization. Their totals and speedup ratios should therefore be interpreted separately. Percentages refer to the profiled total; the indented rows are included in the suffix-forward row, not additional costs. The reported activation memory is not total device memory.

Component	Sequential	IDN `n24-K2`	IDN `n24-K4`	IDN `n12-K2-fwd`
Total	16.07	8.20 (1.96×)	11.58 (1.39×)	13.81 (1.16×)
Embed	0.40 (2.5%)	0.43 (5.3%)	0.40 (3.4%)	0.44 (3.2%)
Sequential prefix	3.47 (21.6%)	3.58 (43.6%)	3.40 (29.4%)	9.49 (68.7%)
Initialization	–	0.16 (2.0%)	0.16 (1.4%)	0.98 (7.1%)
Parallel forward / sequential suffix	12.04 (74.9%)	2.70 (32.9%)	5.14 (44.4%)	2.02 (14.6%)
Residual + norm	1.06 (6.6%)	0.10 (1.2%)	0.19 (1.6%)	0.10 (0.7%)
QKV projection	2.18 (13.5%)	0.23 (2.8%)	0.39 (3.4%)	0.18 (1.3%)
VE gate	1.04 (6.5%)	1.37 (16.8%)	2.65 (22.9%)	0.82 (5.9%)
RoPE + QK norm	3.61 (22.5%)	0.38 (4.7%)	0.74 (6.4%)	0.41 (3.0%)
Attention	0.79 (4.9%)	0.09 (1.1%)	0.18 (1.5%)	0.09 (0.6%)
Output projection	1.13 (7.0%)	0.16 (1.9%)	0.29 (2.5%)	0.14 (1.0%)
MLP	2.24 (14.0%)	0.30 (3.7%)	0.59 (5.0%)	0.23 (1.6%)
IDN correction	–	1.16 (14.1%)	2.30 (19.9%)	0.71 (5.1%)
Logits	0.15 (0.9%)	0.17 (2.1%)	0.17 (1.5%)	0.17 (1.2%)
PPL	15.36	18.09 (+17.8%)	16.42 (+6.9%)	15.59 (+1.5%)
Activation memory	53.8 MB	94.3 MB	94.3 MB	59.4 MB
`bench()`	13.94	8.15	10.77	13.62

Table 22: Component latency and activation memory for Nanochat-0.5B standard (32 layers, with value embeddings). Timing values are milliseconds; percentages are fractions of the profiled total. Indented rows decompose the suffix forward. Parenthesized speedups in the Total row use profiled totals, while `bench()` is timed independently.

For 0.5B `n24-K2`, the suffix forward falls from 12.04 to 2.70 ms, while the sequential prefix, VE gate, and correction cost 3.58, 1.37, and 1.16 ms. Activation memory rises from 53.8 to 94.3 MB. At 3B, the 24-layer prefix already accounts for 66.2% of sequential profiled latency; parallelizing the remaining eight layers reduces the suffix from 4.25 to 1.09 ms at $K = 1$, but the profiled total improves only from 14.54 to 12.20 ms. At $K = 4$, the profiled total exceeds sequential latency. Thus faster suffix evaluation does not by itself guarantee a large end-to-end gain: the sequential prefix, iteration count, and implementation overheads remain consequential. These are measurements of the current implementation; no custom-kernel speedups are claimed.

F.2 FHE-Compatible Approximation Simulation

Fully homomorphic encryption (FHE) provides a different cost regime in which sequential nonlinear depth matters. We simulate degree-12 FHE-compatible nonlinear approximations and use a symbolic CKKS cost model (Cheon et al., 2016). This is a simulation and symbolic-cost study, not an encrypted implementation or an encrypted-runtime benchmark.

For a 32-layer model with an N-layer parallel suffix, the sequential nonlinear-stage count is reduced from 32 to $(32 - N) + K$; all suffix layers are evaluated in parallel within each iteration. This counts stages on the critical path, not total layer evaluations. The additive IDN correction contributes no multiplicative depth in this model. Evaluation uses 200k tokens at sequence length 2048. Approximation-error amplification is the PPL with approximated nonlinearities divided by the same method's PPL with unapproximated nonlinearities. The denominator therefore retains each method's solver configuration, separating the added approximation error from SNLP's existing solver bias.

Component	Sequential	IDN `K1-h0`	IDN `K1-fwd`	IDN `K4-h0`
Total	14.54	12.20 (1.19×)	12.79 (1.14×)	17.76 (0.82×)
Embed	0.39 (2.7%)	0.41 (3.3%)	0.40 (3.1%)	0.44 (2.5%)
Sequential prefix	9.63 (66.2%)	10.13 (83.0%)	9.93 (77.7%)	11.49 (64.7%)
Initialization	–	0.07 (0.6%)	0.91 (7.1%)	0.08 (0.5%)
Parallel forward / sequential suffix	4.25 (29.2%)	1.09 (8.9%)	1.04 (8.1%)	4.47 (25.2%)
Residual + norm	0.34 (2.3%)	0.05 (0.4%)	0.05 (0.4%)	0.21 (1.2%)
QKV projection	0.71 (4.9%)	0.15 (1.2%)	0.13 (1.0%)	0.54 (3.1%)
VE gate	0.32 (2.2%)	0.25 (2.1%)	0.24 (1.9%)	1.09 (6.1%)
RoPE + QK norm	1.14 (7.9%)	0.19 (1.5%)	0.18 (1.4%)	0.83 (4.7%)
Attention	0.26 (1.8%)	0.04 (0.4%)	0.04 (0.3%)	0.18 (1.0%)
Output projection	0.36 (2.5%)	0.09 (0.8%)	0.09 (0.7%)	0.39 (2.2%)
MLP	1.11 (7.6%)	0.28 (2.3%)	0.28 (2.2%)	1.13 (6.4%)
IDN correction	–	0.21 (1.7%)	0.22 (1.7%)	0.97 (5.5%)
Logits	0.28 (1.9%)	0.30 (2.4%)	0.29 (2.3%)	0.30 (1.7%)
PPL	10.07	10.46 (+3.9%)	10.43 (+3.6%)	10.09 (+0.2%)
Activation memory	154.5 MB	168.5 MB	168.5 MB	168.5 MB
`bench()`	13.88	11.59	12.32	16.32

Table 23: Component latency and activation memory for Nanochat-3B standard (32 layers, $N = 8$). Units and conventions follow Table 22; component totals and independent `bench()` timings are distinct measurements.

Architecture	Training	Config.	Seq. amp.	SNLP amp.	Seq. stages seq. → SNLP	Est. bootstraps seq. → SNLP
Nanochat-0.5B standard	Baseline	n8-K4	1.472×	1.462×	32 → 28	53 → 46
Nanochat-0.5B standard	IDN Reg.	n24-K4	1.419×	1.356×	32 → 12	53 → 20
Nanochat-0.5B w/o x0/VE	Baseline	n8-K4	1.466×	1.457×	32 → 28	53 → 46
Nanochat-0.5B w/o x0/VE	IDN Reg.	n24-K4	1.524×	1.499×	32 → 12	53 → 20

Table 24: FHE-compatible approximation simulation on Nanochat-0.5B variants. Amplification is relative to each method's own unapproximated PPL. Sequential nonlinear stages and bootstrap counts are symbolic-model quantities; they are not measured encrypted execution costs.

Approximation-error amplification is lower under SNLP in all four tested variants. The largest reduction is on the standard IDN-regularized model, from $1.419\times$ to $1.356\times$. For $N = 24$, the symbolic bootstrap estimate decreases from 53 to 20, a $2.65\times$ reduction. These observations illustrate potential benefits in a depth-sensitive cost regime; they do not establish encrypted wall-clock speedups or lower absolute PPL across different methods.