# THE CONTEXT-READY TRANSFORMER

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

We introduce the *context-ready transformer*, a decoder architecture that pre-contextualizes each token before it enters the transformer block. During left-to-right generation, a correction network combines the previous position's final block representation with the current token embedding. The token enters the block as its embedding plus this correction, while ordinary causal attention and the standard key-value (KV) cache remain unchanged. The central computation separates training depth from online inference depth. A context-ready model has a $D$-layer block. Parallel training performs $K$ shared block evaluations over the full sequence, giving effective training depth $DK$. Streaming inference processes tokens left to right with one block evaluation per new token and online per-token critical path $D$. A pretrained decoder-only transformer can also be converted by adding a zero-initialized correction feed-forward network (FFN) and fine-tuning. We evaluate validation perplexity (PPL) across widths, depths, block sizes, and two datasets, with comparisons against standard transformers and ablations. In single-run comparisons, a $D{=}5$ model obtains lower PPL than a 12-layer transformer while generating $1.7\times$ faster on an A100. Using $K{=}10$ during training, a single-layer model ($D{=}1$) obtains lower PPL than a 6-layer transformer with a $2.6\times$ inference speedup. Parallel training therefore requires $K$ block evaluations per iteration, whereas streaming inference requires one per generated token. For the checkpoints in Table 4, sequential inference and parallel $K{=}10$ evaluation differ by at most 0.01 PPL. Separately, in a 10-hop pointer-chasing task that requires backpropagation through time (BPTT) across the full correction chain, a $D{=}1$ model solves all 11 evaluated query groups, while standard transformers exhibit staircase-like depth dependence.

## 1 INTRODUCTION

In autoregressive generation, a standard $N$-layer transformer predicts the next token, assigns it a context-free base embedding, and devotes part or all of its $N$ layers to re-contextualize it. This round-trip from context to token ID back to context is an artifact of the architecture, not of the problem.

**Context-ready inference.** The *context-ready transformer* shortens this round-trip. Consider sequential left-to-right generation with positions numbered $1, \ldots, T$. As the model processes token $t$, the block output $z_t$ encodes tokens $1, \ldots, t$. When token $t{+}1$ arrives with embedding $e_{t+1}$, a correction network computes a correction from $z_t$ and $e_{t+1}$. The token enters the block as $e_{t+1}$ plus this correction, which supplies a recurrent path for contextual information before the token enters the transformer layers.

**The training challenge.** This sequential process is exact at inference but inherently serial: each position's correction depends on the previous position's fully computed block output. A classical recurrent neural network (RNN) faces the same issue and uses backpropagation through time (BPTT), which unrolls the recurrence for all $T$ positions and makes the sequential training depth scale with sequence length.

We instead choose a fixed number $K$ of block evaluations during training. Evaluation $k$ processes all $T$ positions in parallel using the correction from evaluation $k{-}1$, then computes the next correction from the resulting block outputs. The final logits come from the $K$th block output and therefore use $K{-}1$ correction updates. The sequential depth of this training procedure is $K$, not $T$, so exact BPTT



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.

has $O(T)$ sequential depth while the parallel correction procedure has $O(K)$ sequential depth. In the language-modeling runs with $D \geq 5$, evaluation at $K{=}5$ and $K{=}10$ differs by at most 0.01 PPL (Table 4).

**Depth accounting.** A standard transformer has training depth $N$ and online per-token critical path $N$. A context-ready transformer evaluates its $D$-layer block during each of the $K$ block evaluations, while online generation runs the $D$-layer block once for each new token. Thus $K$ controls the parallel training procedure, while $D$ is the deployed transformer depth. We use $DK$ versus $N$ as an empirical guide: increasing $K$ should help most when $DK < N$, although $K$ evaluations of one shared $D$-layer block are not equivalent to $N$ distinct transformer layers.

**Why training evaluates the block $K$ times and inference once per token.** Two design choices make this possible.

- **Non-cumulative correction.** Each iteration computes a correction from scratch rather than building on the previous one. The iteration takes the form $x^{(k)} = e + F_\theta(x^{(k-1)}, e)$: the base embedding plus a correction that depends on both the previous iterate's block output and the token embedding. Unlike a residual network, which computes $x^{(K)} = e + f(x^{(0)}) + \cdots + f(x^{(K-1)})$—a sum of $K$ corrections—our formulation yields $x^{(K)} = e + F_\theta(x^{(K-1)}, e)$: a single correction. Previous iterations serve only to bring $x^{(K-1)}$ close to the fixed point; once it has converged, additional iterations produce the same output.
- **Past-only contextualization.** The correction at position $t$ depends on two quantities: the block output $z_{t-1}$, which encodes tokens $1, \ldots, t{-}1$, and the token embedding $e_t$. Since $z_{t-1}$ is already cached from processing the previous token, the correction can be computed as soon as token $t$ arrives. We set $z_0 = \mathbf{0}$ for the first token.

Together, these two properties mean that sequential generation produces the fixed-point correction for each new token without iteration (Section 4).

**Left-to-right recurrent transformer computation.** At sequential inference, the correction at position $t$ depends on $z_{t-1}$, which depends on $z_{t-2}$, and so on. This creates a recurrent computation across all $T$ positions. The model does not compress the prefix into a fixed-size recurrent state: it retains ordinary causal attention and the full KV cache. The recurrent dependency runs from the previous final representation to the correction for the next token. The non-cumulative, past-only structure converts this recurrence into a fixed-point problem for parallel training. Instead of exact $T$-step BPTT, training uses $K$ parallel block evaluations over the full sequence. Because the final output uses $K{-}1$ correction updates, positions $K$ and later are not guaranteed to have fixed-point outputs or gradients.

**Experimental evidence.** We evaluate across widths $C \in \{50, \ldots, 2048\}$, depths $D \in \{1, \ldots, 23\}$, block sizes $\{64, 256, 512, 1024\}$, two datasets (OpenWebText and Wikipedia), and a synthetic reasoning task (Section 5). A $D{=}5$, $C{=}1120$ model obtains lower PPL than a 12-layer transformer with $C{=}768$, reduces the number of sequential transformer layers from 12 to 5, and generates $1.7\times$ faster on an A100. With $K{=}10$ training, $D{=}1$, $C{=}2048$ obtains lower PPL than a 6-layer transformer with $C{=}1088$ and generates $2.6\times$ faster. Separately, on 10-hop pointer chasing, $D{=}1$ trained with BPTT solves all 11 evaluated query groups while standard transformers exhibit staircase-like depth dependence. The language-modeling runs require a dedicated correction network to be effective (Section 5.8). A compatible pretrained decoder-only transformer can be converted by adding a zero-initialized correction FFN and fine-tuning.

## 2 Related Work

**Weight-shared and depth-adaptive architectures.** ALBERT (Lan et al., 2020), Universal Transformers (Dehghani et al., 2019) with Adaptive Computation Time (Graves, 2016), Deep Equilibrium Models (Bai et al., 2019), and Huginn (Geiping et al., 2025) share weights across layers or iterations and refine hidden states repeatedly. Context-ready instead uses a dedicated pre-block correction that depends on the preceding position's output and the current token. Parallel training evaluates the shared $D$-layer block $K$ times over all positions. Streaming inference evaluates it once for each new token, using the preceding token's cached output.

**Feedback across positions.** The Feedback Transformer (Fan et al., 2020) forms representations at the current position from higher-level representations at previous positions through feedback memory. Context-ready instead computes one correction from the immediately preceding block output and the current token embedding while retaining standard causal attention and the KV cache. This correction supports parallel training through repeated block evaluations and exact one-pass streaming.

**Early exit and layer skipping.** LayerSkip (Elhoushi et al., 2024), ADEPT (Yoo et al., 2026), and PonderNet (Banino et al., 2021) reduce average computation through early exit or learned stopping. Context-ready uses a fixed depth with no stopping criterion.

**Parallel recurrent training.** Recent work also solves sequence-wide systems to parallelize recurrent models. DEER (Lim et al., 2024) applies a Newton-style fixed-point method to nonlinear sequential models. Gonzalez et al. (2025) relate fixed-point iterations for nonlinear recurrences through linear dynamical systems, while ParaRNN (Danieli et al., 2025) uses Newton iterations to train nonlinear recurrent networks in parallel. Context-ready uses a Jacobi-style training iteration specialized to a past-only replacement correction. Once previous outputs are cached, the same correction has an exact one-pass streaming evaluation.

**Lookahead Decoding** (Fu et al., 2024) and Consistency Large Language Models (Kou et al., 2024) apply Jacobi iteration to standard transformers as a *decoding strategy*. Song et al. (2021) accelerate deep-equilibrium inference by parallelizing fixed-point solves with Jacobi-style updates. Context-ready changes the architecture: each of the $K$ block evaluations produces a Jacobi update of the correction, while the non-cumulative, past-only structure lets one left-to-right streaming pass compute the exact correction.

**Subquadratic and recurrent alternatives.** Mamba (Gu and Dao, 2024), RWKV (Peng et al., 2023), Griffin (De et al., 2024), and xLSTM (Beck et al., 2024) replace full causal self-attention with recurrent or hybrid sequence operators. Context-ready retains full causal self-attention and changes how tokens enter the transformer block. Standard transformers are therefore the appropriate baselines because they retain the same full causal self-attention and KV-cache computation, isolating the change in how tokens enter the block. Systems that replace causal self-attention test a different architectural change.

**Computational complexity of transformers.** Under log-precision and logspace-computable feed-forward networks, fixed-depth transformers can be simulated by uniform $\mathrm{TC}^0$ circuits (Merrill and Sabharwal, 2023). Idealized recurrent neural networks with unbounded numerical precision can simulate Turing machines (Siegelmann and Sontag, 1995), but training a classical RNN on a length-$T$ sequence normally sends gradients through all $T$ recurrent steps. The context-ready transformer at $D = 1$ is recurrent at inference, but its language-modeling runs train with $K$ block evaluations over all positions in parallel rather than by exact $T$-step BPTT.

# 3 METHOD

## 3.1 ARCHITECTURE

We first describe the architecture during sequential left-to-right generation, where context-ready inference is exact. Let positions be numbered $1, \ldots, T$, let $C$ denote the embedding dimension, and let $V$ denote the vocabulary size. The core component is a *$D$-layer block*: $D$ transformer layers, each consisting of causal self-attention and a feed-forward network with residual connections. Let $e_t \in \mathbb{R}^C$ be the learned lookup embedding of token $t$, let $z_{t-1} \in \mathbb{R}^C$ be the cached block output at the preceding position, and set $z_0 = \mathbf{0}$. We write $\texttt{LN}$ for layer normalization.

**Correction.** A dedicated *correction FFN* generates the correction for position $t$ from the cached block output $z_{t-1}$ and the current token embedding $e_t$:

$$\texttt{correction}_t = \texttt{corr_ffn}\big(\texttt{LN}(z_{t-1} + e_t)\big) \tag{1}$$

The correction FFN is Linear($C \to 4C$) $\to$ Gaussian Error Linear Unit (GELU) $\to$ Linear($4C \to C$), with weights separate from the block's FFN. It depends on the previous position's output $z_{t-1}$ and the current token embedding $e_t$. Both inputs are available when token $t$ arrives, so the correction depends on no future tokens. It supplements full causal self-attention and the standard KV cache; it does not compress the prefix.

**Contextualization.** The new token enters the block with the correction added to its raw embedding:

$$\tilde{x}_t = e_t + \texttt{correction}_t \tag{2}$$

**Block processing.** The $D$-layer block applies transformer layers with separate weights and standard residual connections:

$$\begin{aligned} h^{(0)} &= \tilde{x}_t \\ a^{(i)} &= h^{(i-1)} + \texttt{Attn}_i\big(\texttt{LN}_i^a(h^{(i-1)});\ \mathcal{K}_i, \mathcal{V}_i\big), \quad i = 1, \ldots, D \\ h^{(i)} &= a^{(i)} + \texttt{FFN}_i\big(\texttt{LN}_i^f(a^{(i)})\big) \\ z_t &= h^{(D)} \end{aligned} \tag{3}$$

Each $\texttt{Attn}_i$ is causal self-attention with Rotary Position Embeddings (RoPE) (Su et al., 2024). Each $\texttt{FFN}_i$: Linear($C \to 4C$) $\to$ GELU $\to$ Linear($4C \to C$). The parameter $D$ controls inference depth.

**Prediction.**

$$\texttt{logits}_t = W_{\text{head}} \cdot \texttt{LN}_f(z_t), \quad W_{\text{head}} \in \mathbb{R}^{V \times C} \tag{4}$$

After prediction, $z_t$ is cached and the KV caches $\mathcal{K}_i, \mathcal{V}_i$ are updated for future tokens.

### 3.2 Parallel Training

Sequential inference is exact but inherently serial. For training, we evaluate the block $K$ times, processing all $T$ positions in parallel at each step. Gradients flow through $K$ block evaluations rather than $T$ time steps, making the architecture trainable like a transformer despite being recurrent at inference.

Given token embeddings $e = (e_1, \ldots, e_T)$, initialize $\texttt{correction}^{(0)} = \mathbf{0}$. For $k = 1, \ldots, K$, form the full corrected sequence $\tilde{X}^{(k-1)} = (\tilde{x}_1^{(k-1)}, \ldots, \tilde{x}_T^{(k-1)})$ and run the shared $D$-layer causal transformer block on all $T$ positions in parallel. Let $B_\theta$ denote this full-sequence causal block map and write $Z^{(k)} = B_\theta(\tilde{X}^{(k-1)})$, with $Z^{(k)} = (z_1^{(k)}, \ldots, z_T^{(k)})$ and $z_0^{(k)} = \mathbf{0}$. The training iteration is:

$$\begin{aligned} \tilde{x}_t^{(k-1)} &= e_t + \texttt{correction}_t^{(k-1)}, \qquad t = 1, \ldots, T \\ Z^{(k)} &= B_\theta(\tilde{X}^{(k-1)}) \\ \texttt{correction}_t^{(k)} &= \texttt{corr_ffn}\big(\texttt{LN}(z_{t-1}^{(k)} + e_t)\big), \qquad t = 1, \ldots, T. \end{aligned} \tag{5}$$

Here $K$ counts block evaluations. The output $Z^{(K)}$ uses $\texttt{correction}^{(K-1)}$; $\texttt{correction}^{(K)}$ would be used only by another block evaluation. The $D$ transformer layers share weights across iterations $k$ but have separate weights across layers $i$.

**Non-cumulative correction.** Each iteration replaces the previous correction entirely: $\tilde{x}^{(k)} = e + \texttt{correction}^{(k)}$, not $\tilde{x}^{(k)} = \tilde{x}^{(k-1)} + f(\tilde{x}^{(k-1)})$. Only the last correction matters.

**Past-only correction.** The correction at position $t$ uses $z_{t-1}^{(k)}$. Corrections propagate left to right: position 1 is exact after one iteration, position 2 after two, and so on.

**Random-depth training ($k_{\min}$).** Let $K_{\max}$ denote the maximum number of block evaluations. In random-depth runs, we sample a batch depth $K_{\text{batch}}$ from the discrete uniform distribution on $\{k_{\min}, \ldots, K_{\max}\}$, with $k_{\min} = 2$, and compute the loss from $Z^{(K_{\text{batch}})}$. The loss therefore trains the model at each sampled depth.

**Loss and dropout.** In fixed-depth training, the cross-entropy loss is computed from $Z^{(K)}$. Thus parallel $K{=}1$ uses no correction in the predicted representation. Dropout masks are resampled independently at each block evaluation $k$.

### 3.3 Streaming Inference

When a new token arrives with embedding $e_t$, the model computes the correction from the cached $z_{t-1}$ and $e_t$, passes the corrected embedding through the $D$-layer block, caches the output $z_t$, and predicts. This requires one block evaluation rather than the $K$ evaluations used during training.

**Algorithm 1** Context-Ready Streaming Inference

**Require:** Layers $\texttt{Attn}_1, \texttt{FFN}_1, \ldots, \texttt{Attn}_D, \texttt{FFN}_D$; correction FFN; KV caches $(\mathcal{K}_i, \mathcal{V}_i)_{i=1}^{D}$; previous block output $z_{\text{prev}}$ (init. $\mathbf{0}$)

**for** each new token with embedding $e$ **do**
  $\texttt{correction} \leftarrow \texttt{corr_ffn}\big(\texttt{LN}(z_{\text{prev}} + e)\big)$ {from past context + token identity}
  $h \leftarrow e + \texttt{correction}$ {contextualized input}
  **for** $i = 1, \ldots, D$ **do**
    $h \leftarrow h + \texttt{Attn}_i(\texttt{LN}_i^a(h);\ \mathcal{K}_i, \mathcal{V}_i);$  update $\mathcal{K}_i, \mathcal{V}_i$
    $h \leftarrow h + \texttt{FFN}_i(\texttt{LN}_i^f(h))$
  **end for**
  $z_{\text{prev}} \leftarrow h$ {cache for next token's correction}
  $\texttt{logits} \leftarrow W_{\text{head}} \cdot \texttt{LN}_f(h)$
**end for**

**Why inference needs no iteration.** Since earlier positions are already computed and cached, sequential inference determines the correction for token $t$ from $z_{t-1}$ and $e_t$ in one pass (Theorem 2). This computation is exact when the prefix was processed sequentially or otherwise brought to the fixed point. After finite-$K$ parallel prefill, streaming instead continues from the resulting approximate cache. After $K$ correction updates, $\texttt{correction}^{(K)}$ is exact at the first $K$ positions (Lemma 1 in Appendix A.2). The training output $Z^{(K)}$ uses $\texttt{correction}^{(K-1)}$, so its guaranteed exact prefix contains the first $K-1$ positions. At later positions, the correction error shrinks geometrically with the number of updates when the correction operator is contractive (Theorem 3).

## 4 Theoretical Analysis

Full formal statements and proofs are in Appendix A. The fixed-point and finite-step results below concern a fixed deterministic operator with dropout disabled. Training resamples dropout masks across block evaluations.

**Theorem 1** (One-step streaming structure)**.** *For the past-only replacement form $\tilde{x}_t = e_t + \texttt{correction}_t$, once $z_{t-1}$ is available from the previous time step, the correction for token $t$ is computed once. A cumulative residual refinement has no such guarantee unless its residual term algebraically cancels the current iterate and reduces to a replacement map. For a fixed correction operator, the past-only replacement system has a unique left-to-right solution, and streaming computes it exactly.*

Full proof in Appendix A.1.

**Theorem 2** (Exact streaming fixed point)**.** *During sequential generation, the correction at position $t$ uses the already computed and cached output $z_{t-1}$. By prefix consistency (appending tokens does not change the operator at earlier positions, which holds by causal masking), the correction is exact in a single pass.*

Full proof in Appendix A.2.

**Theorem 3** (Training convergence)**.** *If the correction operator $G$ is $L$-Lipschitz with $L < 1$, then $K$ correction updates reduce the correction error to the fixed point by a factor of $L^K$. Under the training convention above, $Z^{(K)}$ uses $\texttt{correction}^{(K-1)}$. This is a sufficient condition for the parallel training approximation, not a condition needed for exact streaming.*

Full formal statement in Appendix A.3.

**Proposition 1** (Depth separation)**.** *Under a stylized state-tracking abstraction (Appendix A.4):*

*(a)* ***Context-ready state error.*** *If the streaming state update is nonexpansive with constant $L_h \leq 1$, starts from the projected initial state, and has one-step error $\varepsilon_D$, its state error is at most $t\varepsilon_D$. If $L_h < 1$, the uniform bound is $\varepsilon_D/(1 - L_h)$.*

*(b)* ***Standard transformers need depth for propagation.*** *With attention window $W$, a standard transformer needs at least $\lceil (T-1)/W \rceil$ layers for information from position 1 to reach position $T$.*

Table 1: At similar streaming linear-map cost, $D$=5, $C$=1120 obtains lower PPL than $N$=12, $C$=768. In the larger comparison group, $D$=6, $C$=2048 approximately matches $N$=12, $C$=1536, while $N$=24, $C$=1088 obtains lower PPL. The context-ready models use fewer online Transformer layers than the compared 12- and 24-layer transformers. The two comparison groups use 100K and 200K training iterations, respectively. All context-ready models use the add variant, $K$=5, and $k_{\min}$=2.

| **Model** | **Linear MACs/tok** | $C/N$ | **Val PPL** |
|---|---|---|---|
| D=5 $C$=1120 | 121M | 224 | **36.38** |
| D=6 $C$=1024 | 117M | 171 | 36.56 |
| Roformer $N$=6 $C$=1088 | 120M | 181 | 37.76 |
| Roformer $N$=12 $C$=768 | 110M | 64 | 37.83 |
| Roformer $N$=2 $C$=1888 | 146M | 944 | 42.99 |
| Roformer $N$=24 $C$=1088 | 376M | 45 | **28.68** |
| Roformer $N$=12 $C$=1536 | 389M | 128 | 29.01 |
| D=6 $C$=2048 | 401M | 341 | 29.04 |
| Roformer $N$=6 $C$=2176 | 411M | 363 | 30.35 |

Full statement and proof in Appendix A.4. The first bound is conditional on the stated contraction and approximation assumptions. The second is a receptive-field bound for local attention; it does not apply to full-context attention.

# 5 Experiments

## 5.1 Setup

**Data.** OpenWebText (OWT) (Gokaslan and Cohen, 2019) with byte-pair encoding (BPE, vocabulary 32,000). Context lengths are 64, 256, 512, or 1024 depending on the experiment. Ablations use English Wikipedia (BPE 16k); additional Wikipedia results are in Appendix C.8. All results are validation perplexity (PPL) on held-out splits.

**Training.** AdamW optimizer (Loshchilov and Hutter, 2019), gradient clipping at 1.0. Learning rate $2 \times 10^{-4}$ unless noted. Training depth $K = 5$ with $k_{\min} = 2$ by default; $K = 10$ where noted. Dropout 0.2, FFN expansion factor 4. Full hyperparameters in Appendix C.

**Compute accounting.** Tables report multiply-accumulate operations (MACs) in learned linear maps for one streaming token. The prediction head costs $VC$ MACs. Attention projections and the FFN cost $12C^2$ MACs per transformer layer, and the correction FFN costs $8C^2$. The totals are $12DC^2 + 8C^2 + VC$ for a context-ready model and $12NC^2 + VC$ for a standard transformer. They omit the context-dependent attention products, which add about $2tC$ MACs per layer when the current token attends to $t$ cached tokens. Tables 2 and 3 compare the same width, while Table 1 reports cross-width tradeoffs. We use measured throughput for inference-speed claims (Section 5.4).

We do not present training-compute-matched (iso-FLOP) baselines. Exact BPTT would evaluate the $D$-layer block once per token but would process positions sequentially. The parallel training procedure evaluates the $D$-layer block $K$ times, with all positions processed in parallel during each evaluation. Streaming inference evaluates the block once per generated token. We compare language-modeling quality at matched token budgets and evaluate efficiency at inference through streaming depth, measured generation speed, and KV-cache memory. This directly measures the architecture's trade: training computation is paid once to reduce the recurring cost of generation.

**Baselines.** We use standard transformers with separate weights per layer and RoPE attention (Su et al., 2024). Unless otherwise stated, each result is one run; run-to-run variation is not estimated. We report the very small-width results in Appendix C.2.

## 5.2 Cross-Width Results

Table 1 compares context-ready and standard transformers across several depth-width choices. In the 100K-iteration group with block size 256, context-ready $D$=5 at $C$=1120 obtains 36.38 PPL,

Table 2: At fixed width and similar streaming linear-map cost, the context-ready models match the validation PPL of the closest-cost roformers. Models use $C = 1024$, block size 256, and 200K iterations on OWT.

| Model | Linear MACs | Val PPL |
|---|---|---|
| Roformer $N$=12 | $144C^2$ | 33.41 |
| Roformer $N$=13 | $156C^2$ | 32.82 |
| **D=12 context-ready** | $\mathbf{152C^2}$ | **32.28** |
| Roformer $N$=14 | $168C^2$ | 32.34 |
| Roformer $N$=24 | $288C^2$ | 29.42 |
| **D=23 context-ready** | $\mathbf{284C^2}$ | **28.89** |

Table 3: The $D$=2 context-ready model obtains lower PPL than the $N$=2 roformer at each tested width. Models use block size 64 and Chinchilla-matched token budgets on OWT.

| $C$ | $N$=2 **PPL** | $D$=2 **PPL** | $\Delta$ | **Relative** |
|---|---|---|---|---|
| 256 | 158.83 | 143.57 | −15.26 | 9.6% |
| 512 | 95.48 | 84.69 | −10.79 | 11.3% |
| 1024 | 72.83 | 60.84 | −11.99 | 16.5% |

compared with 37.76 for roformer $N$=6 at $C$=1088 and 37.83 for roformer $N$=12 at $C$=768. In the 200K-iteration group, context-ready $D$=6 at $C$=2048 obtains 29.04, compared with 29.01 for roformer $N$=12 at $C$=1536. At the larger streaming-cost scale, the $N$=24, $C$=1088 standard transformer obtains 28.68 PPL, compared with 29.04 for the $D$=6, $C$=2048 context-ready model. The context-ready model uses one quarter as many online transformer layers and 2.1× less KV-cache memory.

### 5.3 Correction Efficiency

Table 2 compares models at fixed width ($C = 1024$), so the prediction head is identical across rows and the reported streaming linear-map MACs are closely matched. The $D$=12 context-ready model matches the PPL of the closest-cost roformers, and the $D$=23 model obtains lower PPL than $N$=24 at similar streaming linear-map cost. The correction FFN adds $8C^2$ MACs.

### 5.4 Width Scaling

In Table 3, $D$=2 obtains lower PPL than $N$=2 at each tested width. The observed relative differences are 9.6%, 11.3%, and 16.5%. The sweep compares the complete architectures as width and the correction FFN's share of the model change together.

**Token-matched results.** At $C = 1024$ with block size 64, the context-ready model has lower PPL than the same-depth roformer after a crossover point in each run: $D$=1 is lower than $N$=1 by 34.4 PPL (crossover at ∼424M tokens), $D$=2 by 7.6 PPL (∼565M), $D$=3 by 3.0 PPL (∼835M), and $D$=6 by 0.7 PPL (∼1,032M). The observed margin is largest at depth one.

**Inference latency.** We measure batch-one autoregressive generation on an A100 over 10,000 tokens with KV caching. $D$=1, $C$=2048 (149M linear MACs/token) generates at 919 tokens/s, compared with 351 tokens/s for roformer $N$=6, $C$=1088 (120M linear MACs/token), a 2.6× speedup. $D$=5, $C$=1120 (121M linear MACs/token) generates at 349 tokens/s, compared with 201 tokens/s for roformer $N$=12, $C$=768 (110M linear MACs/token), a 1.7× speedup. These measurements compare the complete deployed configurations. Across the tested lengths from 100 to 10,000 tokens, measured per-token latency changes by less than 4%. KV caching avoids recomputing the prefix, although attention work for each new token still grows linearly with the number of cached tokens. Full timing details are in Appendix C.9.

**KV cache savings.** Fewer layers also reduce KV cache memory ($C \times D$ per token). Despite being wider, $D$=5 at $C$=1120 uses 1.6× less cache than $N$=12 at $C$=768; $D$=1 at $C$=2048 uses 3.2× less than $N$=6 at $C$=1088.

### 5.5 SINGLE-LAYER PERFORMANCE ($D$=1)

Proposition 1(a) motivates testing whether context propagation across tokens can reduce the transformer depth used during generation. We test $D$=1, $C$=2048 (149M linear MACs/token at inference) against roformer $N$=6, $C$=1088 (120M linear MACs/token) at block size 1024. Streaming generation evaluates the single-layer block once per token. We use three training strategies: fixed $K$=10, random-depth $K$=10 with $k_{\min}$=2, and fine-tuning from a pretrained $N$=1 roformer (batch 16).

Table A8 compares the three strategies. Increasing the training depth from $K$=5 to $K$=10 lowers PPL. After 100K training iterations for each model, fixed $K$=10 obtains 34.40 PPL, compared with 35.37 for $N$=6. The deployed context-ready configuration generates 2.6× faster and uses 3.2× less KV-cache memory (Section 5.4).

**Random-depth training** ($k_{\min}$=2) reaches the same best reported PPL as fixed-depth training, 33.63, about 25K iterations later.

**Fine-tuning** from a pretrained $N$=1 roformer ($K$=1) with a zero-initialized correction FFN at $K$=10 reaches the best final PPL: 31.35 at 215K total iterations.

The gap between $D$=1 and $N$=6 shrinks with block size at fixed $K$=5 (+1.19 at 256, +0.46 at 512, +0.31 at 1024). Across the tested block sizes from 256 to 1024, increasing context length does not require a larger $K$ to preserve the comparison. Context lengths beyond 1024 are not evaluated. Here $K$ controls the depth of the parallel training approximation, while sequential inference applies the context-ready update once per token. Full block-size scaling is in Appendix C.5.

### 5.6 POINTER CHASING: DEPTH SEPARATION

Proposition 1(b) gives a direct depth limit for local attention: each layer extends the receptive field by at most one attention window. We test this limit with a *pointer-chasing* task. The input contains a base table that maps keys to values, followed by nine index tables that map new keys to keys at the previous level. Resolving a top-level key requires ten table lookups, including the base-table lookup. We use windowed causal attention (window = 38) to prevent the model from bypassing the chain by attending directly to the base table. A full task specification with a worked example is given in Appendix B.

Table A1 shows a staircase-like depth dependence: deeper transformers solve more query groups, while shallow transformers fail before full depth. Fixed-$K$ training does not work for this task because it truncates gradients before they traverse the full correction chain. The context-ready $D$=1 model, trained with BPTT through the recurrent correction chain, solves all 11 query groups in ∼16K iterations. A separate 20-hop run solves all 21 query groups. Pointer chasing tests recurrent capacity under BPTT; the language-modeling experiments train with $K$ block evaluations per iteration.

### 5.7 FINE-TUNING FROM PRETRAINED

A pretrained decoder-only transformer can be converted by adding a zero-initialized correction FFN, preserving its function before fine-tuning. At $C$=1408, the converted model reaches 26.14 PPL after 400K total iterations, compared with 27.20 for the original model continued to 400K; each fine-tuning iteration runs the block $K$ times. Appendix C.7 reports the full results.

### 5.8 ABLATIONS AND DIAGNOSTICS

We test whether the observed difference comes from the dedicated correction network or merely from adding another FFN. Among the tested variants, only the dedicated correction network with separate weights improves over the corresponding baseline.

**Block evaluations and sequential exactness.** Table 4 shows the full depth progression. PPL approaches its $K$=5 value rapidly: at $D$=1 (block size 1024), $K$=2 closes 91% of the $K$=1-to-$K$=5 gap and $K$=3 closes 98%. This behavior is consistent with the geometric error bound in Theorem 3, but it does not estimate the theorem's global contraction constant. For the checkpoints in Table 4, sequential evaluation matches parallel $K$=10 to within 0.01 PPL, and at $D \geq 5$, $K$=5 and $K$=10

Table 4: Additional block evaluations rapidly approach the $K{=}5$ PPL, and parallel $K{=}10$ and sequential evaluation agree within 0.01 PPL for these checkpoints. The $D{=}1$ models use $C{=}2048$; the others use $C{=}1024$ and block size 256. "Corr." is the PPL increase when correction is disabled at parallel $K{=}1$, relative to $K{=}5$.

| $D$ | **bs** | $K{=}1$ | $K{=}2$ | $K{=}3$ | $K{=}5$ | $K{=}10$ | **Seq** | **Corr.** |
|---|---|---|---|---|---|---|---|---|
| 1 | 256 | 70.80 | 36.21 | 33.23 | 32.70 | 32.79 | 32.80 | 38.1 |
| 1 | 512 | 73.22 | 34.33 | 31.24 | 30.61 | 30.68 | 30.69 | 42.6 |
| 1 | 1024 | 84.13 | 34.42 | 30.39 | 29.51 | 29.43 | **29.43** | 54.7 |
| 5 | 256 | 58.17 | 40.18 | 38.80 | 38.60 | 38.61 | 38.62 | 19.6 |
| 12 | 256 | 38.33 | 32.58 | 32.30 | 32.29 | 32.29 | 32.29 | 6.0 |
| 23 | 256 | 32.80 | 29.03 | 28.89 | 28.88 | 28.88 | 28.88 | 3.9 |

differ by at most 0.01 PPL. The former provides a numerical check against Theorem 2. The correction contribution ("Corr." column) is 38–55 PPL at $D{=}1$ and 3.9 at $D{=}23$.

**Correction FFN is essential.** Using the block's own residual as the correction ("block_head") gives no improvement (27.32 vs. 27.19 for a standard transformer). With a dedicated correction FFN, context-ready obtains 1.8 lower PPL than the baseline with a similar linear-map cost. The add variant ($\mathtt{LN}(z_{t-1} + e_t)$) matches or beats token-blind at all depths tested.

## 6 Conclusion

A standard transformer assigns each new token a context-free embedding and relies entirely on depth to contextualize it. The context-ready transformer shortcuts this process: a correction derived from the previous position's block output pre-contextualizes the token before it enters the block. Two structural choices—non-cumulative correction and past-only contextualization—make this exact at streaming inference (Theorem 2) and trainable with $K$ block evaluations, each processing all $T$ positions in parallel, rather than full BPTT. The resulting accounting is explicit: parallel training has effective depth $DK$, while online generation for each next token has critical path $D$.

The $D{=}5$ model obtains lower PPL than a 12-layer transformer while generating $1.7\times$ faster. With $K{=}10$ training, the $D{=}1$ model obtains lower PPL than a 6-layer transformer with a $2.6\times$ inference speedup. Fewer layers also reduce KV cache memory. On 10-hop pointer chasing, $D{=}1$ trained with BPTT solves all 11 query groups, demonstrating recurrent capacity under BPTT. A compatible pretrained decoder-only transformer can be converted by adding a zero-initialized correction FFN and fine-tuning.

**Limitations.** *Evaluation.* The experiments focus on language modeling and synthetic reasoning to isolate the architecture's behavior. Each reported result is one run, so the paper provides no estimate of run-to-run variation. Scaling and downstream evaluation remain future work. *Training.* From-scratch training runs the block $K$ times per iteration and stores activations for all $K$ passes. Fixed-small-$K$ training is evaluated on language modeling, while pointer chasing uses BPTT to evaluate recurrent capacity. Pretraining a standard transformer and then adding the zero-initialized correction reduces the amount of context-ready training. *Serving.* The reported throughput uses batch-one, cross-width comparisons and excludes prompt-processing latency. Finite-$K$ parallel prefill produces an approximate cache with depth $KD$ and may require more prompt-processing computation and time to first token than depth $N$.

## AI Assistance Disclosure

Generative AI tools were used to polish and occasionally draft manuscript language, locate related work, and assist in checking and revising mathematical arguments and proofs. The author originated the architecture, theoretical framework, and mathematical claims, designed and ran the experiments, interpreted the results, and made all final decisions. The author reviewed and verified all AI-assisted material and takes responsibility for the paper.

## REFERENCES


Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/01386bd6d8e091c2ab4c7c7de644d37b-Abstract.html.

Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021. URL https://arxiv.org/abs/2107.05407.

Maximilian Beck, Korbinian Pöppel, Markus Spannring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xLSTM: Extended long short-term memory. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-3417. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/c2ce2f2701c10a2b2f2ea0bfa43cfaa3-Abstract-Conference.html.

Federico Danieli, Pau Rodriguez, Miguel Sarabia, Xavier Suau, and Luca Zappella. ParaRNN: Unlocking parallel training of nonlinear RNNs for large language models. *arXiv preprint arXiv:2510.21450*, 2025. URL https://arxiv.org/abs/2510.21450.

Soham De, Samuel L. Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, Guillaume Desjardins, Arnaud Doucet, David Budden, Yee Whye Teh, Razvan Pascanu, Nando De Freitas, and Caglar Gulcehre. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024. URL https://arxiv.org/abs/2402.19427.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12622–12642. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.681. URL https://aclanthology.org/2024.acl-long.681/.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing some limitations of transformers with feedback memory. *arXiv preprint arXiv:2002.09402*, 2020. URL https://arxiv.org/abs/2002.09402.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 14060–14079. PMLR, 2024. URL https://proceedings.mlr.press/v235/fu24a.html.

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. *arXiv preprint arXiv:2502.05171*, 2025. URL https://arxiv.org/abs/2502.05171.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. https://skylion007.github.io/OpenWebTextCorpus/, 2019.

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. *arXiv preprint arXiv:2509.21716*, 2025. URL https://arxiv.org/abs/2509.21716.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016. URL https://arxiv.org/abs/1603.08983.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

Siqi Kou, Lanxiang Hu, Zhezhi He, Zhijie Deng, and Hao Zhang. CLLMs: Consistency large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 25426–25440. PMLR, 2024. URL https://proceedings.mlr.press/v235/kou24a.html.

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. URL https://openreview.net/forum?id=H1eA7AEtvS.

Yi Heng Lim, Qi Zhu, Joshua Selfridge, and Muhammad Firmansyah Kasim. Parallelizing non-linear sequential models over the sequence length. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=E34AlVLN0v.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023. doi: 10.1162/tacl_a_00562. URL https://aclanthology.org/2023.tacl-1.31/.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Leon Derczynski, Xingjian Du, Matteo Grella, Kranthi Gv, Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartłomiej Koptyra, Hayden Lau, Jiaju Lin, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Johan Wind, Stanisław Woźniak, Zhenyuan Zhang, Qinghua Zhou, Jian Zhu, and Rui-Jie Zhu. RWKV: Reinventing RNNs for the transformer era. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14048–14077. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-emnlp.936. URL https://aclanthology.org/2023.findings-emnlp.936/.

Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995. doi: 10.1006/jcss.1995.1013.

Yang Song, Chenlin Meng, Renjie Liao, and Stefano Ermon. Accelerating feedforward computation via parallel nonlinear equation solving. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9791–9800. PMLR, 2021. URL https://proceedings.mlr.press/v139/song21a.html.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. doi: 10.1016/j.neucom.2023.127063.

Sangmin Yoo, Srikanth Malla, Chiho Choi, Wei D. Lu, and Joon Hee Choi. ADEPT: Adaptive dynamic early-exit process for transformers. *arXiv preprint arXiv:2601.03700*, 2026. URL https://arxiv.org/abs/2601.03700.

# A FULL PROOFS

**Notation.** Throughout the appendix, $e$ denotes the base token embeddings, $G$ denotes the full correction operator, and $c^{(k)}$ denotes the correction vector at iteration $k$. The fixed point is $c^* = G(c^*)$.

## A.1 PROOF OF THEOREM 1 (ONE-STEP STREAMING STRUCTURE)

**Formal statement.**

*Setup.* Let $e_t \in \mathbb{R}^C$ denote the base embedding at position $t$, let `Block` denote a $D$-layer transformer block with causal attention, and let $F : \mathbb{R}^C \times \mathbb{R}^C \to \mathbb{R}^C$ be a correction function. The architecture processes position $t$ as follows: form the corrected input $\tilde{x}_t = e_t + F(e_t, z_{t-1})$, compute $z_t = \texttt{Block}(\tilde{x}_t;\, \tilde{x}_{<t})$, and cache $z_t$ for future positions.

*Design properties.*

(I) *Replacement correction.* The corrected input has the form $\tilde{x}_t = e_t + F(e_t, z_{t-1})$. Each evaluation replaces the previous correction rather than adding another residual update.

(II) *Single-pass streaming.* During inference, tokens are processed left-to-right, one at a time. The correction at position $t$ uses the cached block output $z_{t-1}$ from the previous position and the current embedding $e_t$. By causality, $z_{t-1}$ depends only on positions $\leq t-1$.

*Conclusions.*

(a) *Past-only.* By streaming (Property II), the correction at position $t$ depends only on previously computed outputs: $F(e_t, z_{t-1})$, where $z_{t-1}$ encodes positions $1, \ldots, t-1$.

(b) *Replacement versus cumulative residual refinement.* The non-cumulative form $\tilde{x}_t^{(k)} = e_t + F(e_t, z_{t-1}^{(k)})$ gives a structural one-step streaming rule once $z_{t-1}$ is available. A cumulative residual alternative $\tilde{x}_t^{(k)} = \tilde{x}_t^{(k-1)} + H(\tilde{x}_t^{(k-1)}, z_{t-1}^{(k)})$ does not give this guarantee unless $H$ algebraically cancels the current iterate and thereby reduces to a replacement map.

(c) *Unique left-to-right solution.* For a fixed past-only correction operator, the non-cumulative system determines a unique left-to-right solution, and streaming computes it exactly.

*Proof.* **Part (a): Past-only.** Streaming (Property II) processes tokens left-to-right. When token $t$ arrives, the correction uses $z_{t-1}$ (cached from the previous token) and $e_t$. Since causal attention ensures $z_{t-1}$ depends only on positions $\leq t-1$, the correction at position $t$ is a function of past outputs only.

**Part (b): Replacement versus cumulative residual refinement.** Given the replacement correction in Property I, there are two ways to unroll $K$ training steps:

*Non-cumulative:* $\tilde{x}_t^{(k)} = e_t + F(e_t, z_{t-1}^{(k)})$. Each step recomputes the correction from the base embedding $e_t$. At the fixed point, $F(e_t, z_{t-1}^*) = c_t^*$. In streaming, past outputs are exact from the cache, so a single evaluation gives $\tilde{x}_t = e_t + F(e_t, z_{t-1}^*) = e_t + c_t^* = \tilde{x}_t^*$. Thus one streaming evaluation is exact.

*Cumulative residual:* $\tilde{x}_t^{(k)} = \tilde{x}_t^{(k-1)} + H(\tilde{x}_t^{(k-1)}, z_{t-1}^{(k)})$. Each step adds an increment to the previous output. At the fixed point,

$$\tilde{x}_t^* = \tilde{x}_t^* + H(\tilde{x}_t^*, z_{t-1}^*),$$

so

$$H(\tilde{x}_t^*, z_{t-1}^*) = 0.$$

This condition alone does not structurally determine a one-step streaming rule from the base embedding $e_t$: exact one-step streaming would additionally require

$$e_t + H(e_t, z_{t-1}^*) = \tilde{x}_t^*.$$

Equivalently,

$$H(e_t, z_{t-1}^*) = \tilde{x}_t^* - e_t$$

while also satisfying $H(\tilde{x}_t^*, z_{t-1}^*) = 0$. A syntactically cumulative update can satisfy both equations, but only by behaving like a replacement map on the relevant states. For example, if

$$H(x, z) = h(z) - x,$$

then

$$x + H(x, z) = h(z),$$

so the residual term cancels the current iterate and the cumulative-looking update is exactly a replacement/projection update written in residual notation.

Therefore the design claim is not that a syntactically cumulative formula is logically impossible. The claim is that ordinary cumulative residual refinement provides no structural guarantee of one-step streaming unless it collapses algebraically to replacement. Context-ready builds the replacement form directly.

**Part (c): Unique left-to-right solution.** By part (a), the system is triangular: $\tilde{x}_1^*$ is determined first (from $z_0 = \mathbf{0}$ and $e_1$), then $z_1^*$, then $\tilde{x}_2^*$, and so on. For a fixed correction operator and fixed block, each step is a deterministic evaluation. Hence the left-to-right sequence is unique and is constructively computable by streaming. □

### A.2 Proof of Theorem 2 (Exact Streaming)

**Formal statement.** Let $c^{*(T)}$ be the unique fixed point for a sequence of length $T$. Assume *prefix consistency*: for any $T' > T$ and $t \leq T$, the correction operator satisfies $G_t^{(T')} = G_t^{(T)}$, i.e., appending tokens beyond position $T$ does not change the operator at earlier positions. (This holds by construction for any causal architecture where the operator at position $t$ depends only on positions $\leq t$.) Then:

(a) *Prefix invariance.* $c_t^{*(T')} = c_t^{*(T)}$ for $t \leq T$ and any $T' > T$.

(b) *Exactness.* If past corrections are at the fixed point, the streaming operator produces the exact fixed-point correction for the new token.

(c) *No contraction needed.* Exactness holds without any contraction assumption.

The following lemma gives the parallel counterpart of streaming exactness: each Jacobi iteration makes one additional prefix position exact.

**Lemma 1** (Finite-step exactness)**.** *Let $G$ be a past-only correction operator. Then:* (a) *The fixed point $c^*$ exists and is unique, without contraction.* (b) *After $k$ Jacobi iterations from any $c^{(0)}$: $c_t^{(k)} = c_t^*$ for all $t \leq k$.* (c) *The iteration reaches the exact fixed point after at most $T$ steps: $c^{(T)} = c^*$.*

*Proof.* (a) By construction: $G_1$ is a constant, so $c_1^*$ is determined. Given $c_1^*, \ldots, c_{t-1}^*$, we have $c_t^* = G_t(c_1^*, \ldots, c_{t-1}^*)$ uniquely.

(b) By induction. Base: $c_1^{(1)} = G_1 = c_1^*$. Step: assume $c_s^{(k)} = c_s^*$ for $s \leq k$. For each $s \leq k$, past-only dependence gives $c_s^{(k+1)} = G_s(c_{<s}^{(k)}) = G_s(c_{<s}^*) = c_s^*$. The next position also becomes exact:

$$c_{k+1}^{(k+1)} = G_{k+1}(c_1^{(k)}, \ldots, c_k^{(k)}) = G_{k+1}(c_1^*, \ldots, c_k^*) = c_{k+1}^*.$$

(c) Set $k = T$ in (b). □

*Proof of Theorem 2.* (a) By prefix consistency, $G_t^{(T')} = G_t^{(T)}$ for $t \leq T$. Hence the fixed-point equations for positions $1, \ldots, T$ are identical in the length-$T$ and length-$T'$ systems, so $c_t^{*(T')} = c_t^{*(T)}$.

(b) $c_{T+1}^{*(T+1)} = G_{T+1}^{(T+1)}(c_1^{*(T+1)}, \ldots, c_T^{*(T+1)})$. By (a), $c_t^{*(T+1)} = c_t^{*(T)}$ for $t \leq T$. The streaming operator computes exactly $G_{T+1}^{(T+1)}$ using cached corrections $c^{*(T)}$. Concretely, the abstract operator $G_t$ is realized by the correction FFN applied to the cached block output $z_{t-1}$ and the current token

embedding $e_t$: once earlier positions are at their fixed-point values, $z_{t-1}$ is fully determined, and the streaming step computes $c_t = G_t(c_1^*, \ldots, c_{t-1}^*; e_t)$ exactly.

(c) By induction from the base case and part (b). □

### A.3 Proof of Theorem 3 (Convergence)

**Formal statement.** Let $G_t(c_{<t}; e)$ denote the full correction operator at position $t$. Assume $\|G(c; e) - G(c'; e')\| \le L\|c - c'\| + M\|e - e'\|$ for all $c, c', e, e'$. If $L < 1$, then:

(a) *Contraction.* $\|c^{(k)} - c^*\| \le L^k \|c^{(0)} - c^*\|$.

(b) *Warm-start bound.* $\|G(c^*; e') - c^{*\prime}\| \le \frac{LM}{1-L}\|e' - e\|$.

*Proof.* **Part (a).** By the Banach fixed-point theorem with contraction constant $L < 1$.

**Part (b).** $\|c^* - c^{*\prime}\| \le M\|e - e'\| + L\|c^* - c^{*\prime}\|$, so $\|c^* - c^{*\prime}\| \le \frac{M}{1-L}\|e' - e\|$. Then $\|G(c^*; e') - c^{*\prime}\| \le L\|c^* - c^{*\prime}\| \le \frac{LM}{1-L}\|e' - e\|$. □

Theorem 3 assumes $L < 1$ but does not say how to verify this from the per-position Jacobian structure. The following lemma provides a practical bound: given bounds on the partial derivatives $\|\partial G_t / \partial c_s\|$, one can choose a weighted norm that makes the global Lipschitz constant explicit.

**Lemma 2** (Causal contraction bound)**.** *Let $G(c; e)$ be the full-sequence correction operator with past-only dependencies, and suppose $\|\partial G_t / \partial c_s\|_{\text{op}} \le a_{t,s}$ for $s < t$, where $\|A\|_{\text{op}} = \sup_{\|x\|=1} \|Ax\|$ is the operator norm. For positive weights $w = (w_1, \ldots, w_T)$, define $\|c\|_w = \max_t w_t \|c_t\|$. Then $G$ is $L_w$-Lipschitz in $\|\cdot\|_w$ with constant:*

$$L_w = \max_{1 \le t \le T} w_t \sum_{s=1}^{t-1} \frac{a_{t,s}}{w_s}.$$

*In particular, if $L_w < 1$, then $G$ is a contraction.*

*Proof.* For each $t$: $\|G_t(c) - G_t(c')\| \le \sum_{s<t} a_{t,s}\|c_s - c'_s\| \le \sum_{s<t} \frac{a_{t,s}}{w_s}\|c - c'\|_w$. Multiplying by $w_t$ and taking the maximum over $t$ gives $\|G(c) - G(c')\|_w \le L_w \|c - c'\|_w$. □

When $K$ is chosen at training time, one may want to know how close $c^{(K)}$ is to $c^*$ without computing $c^*$. The next lemma gives a computable bound using only consecutive iterates.

**Lemma 3** (A posteriori error bound)**.** *If $L < 1$, then after $k$ iterations: $\|c^{(k)} - c^*\| \le \frac{L}{1-L}\|c^{(k)} - c^{(k-1)}\|$.*

*Proof.* By the triangle inequality, $\|c^{(k)} - c^*\| \le \sum_{j=1}^{\infty} \|c^{(k+j)} - c^{(k+j-1)}\| \le \sum_{j=1}^{\infty} L^j \|c^{(k)} - c^{(k-1)}\| = \frac{L}{1-L}\|c^{(k)} - c^{(k-1)}\|$. □

Theorem 3(a) gives a global contraction rate $L^k$ that treats all positions uniformly, but this is overly pessimistic. Because $G$ is past-only, its Jacobian is strictly lower-triangular and therefore nilpotent, so position $t$ reaches its exact fixed point after at most $t$ iterations rather than $T$. The following lemma exploits this structure to give a tighter, position-dependent error bound via causal path sums.

**Lemma 4** (Finite-depth error bound)**.** *Let $G$ be a past-only correction operator with Jacobian bounds $a_{t,s}$. Let $A$ be the strictly lower-triangular matrix with entries $a_{t,s}$, and $B = \max_t \|G_t(0)\|$. After $N$ Jacobi iterations from $c^{(0)} = 0$:*

$$\|c_t^{(N)} - c_t^*\| \le \sum_{k=N}^{T-1} [A^k]_{t,\cdot} \mathbf{1} \cdot B = [(A^N (I - A)^{-1})_{t,\cdot}] \mathbf{1} \cdot B$$

*Proof.* Let $|c^*| \in \mathbb{R}^T$ denote the vector with entries $|c^*|_t = \|c_t^*\|$. By the integral form of the mean value theorem, the error $e^{(k)} = c^{(k)} - c^*$ satisfies $e_t^{(k+1)} = \sum_{s<t} J_{t,s}^{(k)} e_s^{(k)}$ where $\|J_{t,s}^{(k)}\| \leq a_{t,s}$ by the Jacobian bound assumption. Bounding by the entrywise matrix $A$ and iterating from $e^{(0)} = -c^*$ gives $\|e_t^{(N)}\| \leq [A^N|c^*|]_t$. To bound $|c^*|$, the fixed point satisfies $c_t^* = G_t(c_{<t}^*)$, so $\|c_t^*\| \leq \|G_t(0)\| + \sum_{s<t} a_{t,s}\|c_s^*\|$. Therefore $|c^*| \leq B \cdot \mathbf{1} + A|c^*|$ and $|c^*| \leq (I-A)^{-1}B \cdot \mathbf{1}$, which is well-defined because $A$ is nilpotent. Substitution gives $\|e_t^{(N)}\| \leq [A^N(I-A)^{-1}]_{t,\cdot} \mathbf{1} \cdot B$. □

### A.4 Proof of Proposition 1 (Depth Separation)

**Formal statement.**

*Setup.* Suppose the data is generated by a process with state update $\mathbf{s}_t = h^*(\mathbf{s}_{t-1}, \mathbf{e}_t)$, where $\mathbf{s}_t \in \mathbb{R}^m$ is the process state and $\mathbf{e}_t \in \mathbb{R}^C$ is the current token embedding. The context-ready architecture (Equations 1–3) with attention window $W$ maintains a state $\hat{\mathbf{b}}_t \in \mathbb{R}^{nW}$ containing the cached representations for the current window, where $n$ is the per-position state dimension. The state evolves via $\hat{\mathbf{b}}_t = h(\hat{\mathbf{b}}_{t-1}, \mathbf{e}_t)$, where $h : \mathbb{R}^{nW} \times \mathbb{R}^C \to \mathbb{R}^{nW}$ composes the correction FFN, the block, and the cache update. Let $L_h$ be the Lipschitz constant of $h$ in its first argument.

To compare these two systems, let $\pi : \mathbb{R}^m \to \mathbb{R}^{nW}$ map the process state to the architecture's state space. Define the one-step error

$$\varepsilon_D \;=\; \sup_{\mathbf{s},\,\mathbf{e}} \big\| h\big(\pi(\mathbf{s}),\, \mathbf{e}\big) - \pi\big(h^*(\mathbf{s},\, \mathbf{e})\big) \big\|$$

This quantity measures the difference between one architecture update and the corresponding process update. It includes the block's finite-depth approximation error and any information lost when $\pi$ maps between the two state spaces.

*Assumptions.*

(i) $L_h \leq 1$, $\varepsilon_D < \infty$, and $\hat{\mathbf{b}}_0 = \pi(\mathbf{s}_0)$.

(ii) *Prediction sufficiency.* $\pi$ preserves prediction-relevant information: $p(x_{t+1} \mid \mathbf{s}_t) = p(x_{t+1} \mid \pi(\mathbf{s}_t))$.

(iii) *Lipschitz readout.* The prediction function $\phi : \mathbb{R}^{nW} \to \Delta$ is $L_\phi$-Lipschitz.

*Conclusions.*

(a) *Context-ready error bound.* The accumulated state error satisfies

$$\|\hat{\mathbf{b}}_t - \pi(\mathbf{s}_t)\| \leq \varepsilon_D \sum_{j=0}^{t-1} L_h^j.$$

For $L_h < 1$, this is at most $\varepsilon_D/(1-L_h)$ uniformly in $t$; for $L_h = 1$, it is at most $t\varepsilon_D$. By prediction sufficiency and Lipschitz readout, the corresponding prediction-error bounds are multiplied by $L_\phi$. If $\varepsilon_D = 0$, then streaming is exact for all $t$.

(b) *Standard transformer receptive-field bound.* Let $N$ be the number of layers in a standard transformer with attention window $W$. Then position $t$ has no computational path to any position before $t - NW$, so the transformer cannot represent any function whose output at position $t$ depends on inputs before $t - NW$.

*Proof.* **Part (a).** At step $t$, the architecture computes $\hat{\mathbf{b}}_t = h(\hat{\mathbf{b}}_{t-1}, \mathbf{e}_t)$ while the projected true state satisfies $\pi(\mathbf{s}_t) = \pi(h^*(\mathbf{s}_{t-1}, \mathbf{e}_t))$. By the triangle inequality:

$$\|\hat{\mathbf{b}}_t - \pi(\mathbf{s}_t)\| \;\leq\; \underbrace{\|h(\hat{\mathbf{b}}_{t-1}, \mathbf{e}_t) - h(\pi(\mathbf{s}_{t-1}), \mathbf{e}_t)\|}_{\leq\, L_h \|\hat{\mathbf{b}}_{t-1} - \pi(\mathbf{s}_{t-1})\|} \;+\; \underbrace{\|h(\pi(\mathbf{s}_{t-1}), \mathbf{e}_t) - \pi(h^*(\mathbf{s}_{t-1}, \mathbf{e}_t))\|}_{\leq\, \varepsilon_D}.$$

Using $\hat{\mathbf{b}}_0 = \pi(\mathbf{s}_0)$ and unrolling gives

$$\|\hat{\mathbf{b}}_t - \pi(\mathbf{s}_t)\| \leq \varepsilon_D \sum_{j=0}^{t-1} L_h^j.$$

When $L_h < 1$, the geometric sum is at most $1/(1 - L_h)$. When $L_h = 1$, it equals $t$. The Lipschitz readout gives the stated prediction-error bounds.

**Part (b).** With attention window $W$, the output of layer $\ell$ at position $t$ depends only on positions in $[t - \ell W,\ t]$. After $N$ layers, position $t$ has no computational path to any position before $t - NW$. □

*Remark* (Depth separation). Parts (a) and (b) identify different ways to carry information across tokens. During sequential inference, the context-ready architecture carries state through its recurrent update without increasing the block depth. A standard transformer with local attention must instead increase its layer count or attention window to connect positions separated by more than $NW$. This comparison does not by itself imply an unconditional layer advantage, because layers may perform both propagation and local computation.

## B Pointer Chasing Details

**Motivation.** We design a synthetic task that tests the local-attention depth bound in Proposition 1(b). Answering a query requires chaining a variable number of sequential lookups. A standard transformer must propagate the required information through its finite stack, while the context-ready architecture can propagate it through the recurrent correction chain at inference, even with a single block ($D$=1).

**Task definition.** A task with $H$ table lookups and $M$ keys per level contains a base table (level 0) that maps $M$ keys to values, followed by $H-1$ index tables (levels $1, \ldots, H-1$). Each index table maps $M$ new keys to keys at the previous level through a random permutation. After each table, a query section contains a `Q key` pair for every key introduced at that level. The resolved answer is a supervision target at the key position and is not included in the input. A query at index level $\ell$ requires $\ell$ index-table lookups followed by one base-table lookup.

**Worked example** ($H$=3 table lookups, $M$=3, 10 values). The base table maps A→v3, B→v0, C→v8. Index table 1 maps D→A, E→B, F→C. Index table 2 maps G→D, H→E, I→F. The encoding uses reversed triplets (`value=key`) so that causal attention can see the value to the left of the key:

**Level 0 (base table + queries):**

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Input: | v3 | = | A | v0 | = | B | v8 | = | C | \| |
| Target: | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ |
| Input: | Q | A | Q | B | Q | C | \| | | | |
| Target: | _ | **v3** | _ | **v0** | _ | **v8** | _ | | | |

**Level 1 (index table + queries):**

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Input: | A | = | D | B | = | E | C | = | F | \| |
| Target: | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ |
| Input: | Q | D | Q | E | Q | F | \| | | | |
| Target: | _ | **v3** | _ | **v0** | _ | **v8** | _ | | | |

**Level 2 (index table + final query):**

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Input: | D | = | G | E | = | H | F | = | I | \| | Q | G |
| Target: | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | _ | **v3** |

Targets (bold) appear only at key positions in query sections. Answer tokens are never part of the input. Level-0 queries are direct lookups (Q A → v3). Level-1 queries require one composition (Q D → A → v3). Level-2 queries require two compositions (Q G → D → A → v3). The final token is the test query. Dense targets supervise every intermediate composition level during BPTT.

Each level uses its own key namespace (A, B, C at level 0; D, E, F at level 1; G, H, I at level 2; etc.) to prevent ambiguity. Key ordering within each table is fixed (not shuffled), so the model can exploit positional patterns via RoPE.

**Settings.** $H = 10$ table lookups, $M = 5$ keys, 10 values, embedding dimension $C = 256$, 4 attention heads, batch size 64, window size 38, fixed key ordering, per-level key tokens, RoPE attention. Learning rate $1 \times 10^{-4}$.

**Why windowed attention.** Without windowed attention, all models—including deep transformers—can directly attend from any query position to the base table, achieving $\sim 1/M$ accuracy without

Table A1: With windowed attention, roformers solve more query groups as their depth increases, while the context-ready $D$=1 model trained with BPTT solves all 11 groups. The task uses 10 table lookups, 5 keys, and 10 values.

| **Model** | **Query groups solved** | **Iters** |
|---|---|---|
| Roformer $N$=1 | 1 / 11 | 50K |
| Roformer $N$=3 | 3 / 11 | 50K |
| Roformer $N$=5 | 6 / 11 | 50K |
| Roformer $N$=10 | 7 / 11 | 50K |
| Roformer $N$=11 | 8 / 11 | 50K |
| Roformer $N$=12 | 11 / 11 | 50K |
| **D=1 context-ready (BPTT)** | **11 / 11** | **∼16K** |

genuine composition. Windowed attention ($w = 38$) ensures that higher-level query sections cannot see the base table, forcing the model to chain through intermediate levels. This reveals the true depth-limited structure of fixed-depth transformers.

**Order in which levels are learned.** In the reported $D$=1 run, accuracy rises first at level 0 and then at levels 1, 2, and later levels. The table below reports this progression.

**BPTT progression.** The progression below uses a smaller configuration ($C$=128, lr = 1e-3, 20 values) to show the order in which levels are learned at reduced compute:

Table A2: In the reduced-compute $D$=1 BPTT run, accuracy reaches earlier query groups before later groups.

| **Iter** | L0 | L1 | L2 | L3 | L4 | L5 | L6 | L7 | L8 | L9 | L10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3K | 1.00 | 0.74 | 0.27 | 0.20 | 0.19 | 0.18 | 0.18 | 0.18 | 0.17 | 0.16 | 0.16 |
| 8K | 1.00 | 1.00 | 0.80 | 0.59 | 0.36 | 0.21 | 0.21 | 0.19 | 0.20 | 0.18 | 0.15 |
| 13.5K | 1.00 | 1.00 | 0.99 | 0.99 | 0.94 | 0.83 | 0.51 | 0.24 | 0.20 | 0.20 | 0.18 |
| 23K | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.99 | 0.99 | 0.99 | 0.99 | 0.98 | 0.98 |

**20-hop run.** At $C = 512$ with lr = 1e-4, the same $D$=1 architecture solves all 21 query groups in ∼4K iterations.

# C Extended Experimental Details

## C.1 Hyperparameters

Table A3: Hyperparameters by experiment.

| **Hyperparameter** | ∼85M | ∼340M | Token-matched | Width scaling |
|---|---|---|---|---|
| Block size | 256 | 256 | 64 | 64 |
| Batch size | 64 | 32 | 1024 | 1024 |
| Learning rate | $2 \times 10^{-4}$ | $2 \times 10^{-4}$ | $2 \times 10^{-4}$ | $2 \times 10^{-4}$ |
| Training iters | 100K | 200K | varies | varies |
| $K$ / $k_{\min}$ | 5 / 2 | 5 / 2 | 5 / 2 | 5 / 2 |
| Dropout | 0.2 | 0.2 | 0.2 | 0.2 |
| Vocab size | 32,000 | 32,000 | 32,000 | 32,000 |

### C.2 Ablations

Table A4: A dedicated correction FFN improves on the matched-cost roformer, while reusing the block residual does not. Models use $C = 446$ on Wikipedia with BPE 16k. "Roformer-hFFN" adds an FFN to match the correction FFN's cost.

| Model | Linear MACs/tok | Val PPL | Seq. $K{=}1$ |
|---|---|---|---|
| D=3 corr_ffn $K{=}5$ | $44C^2$ | **23.98** | 23.96 |
| Roformer-hFFN $N{=}3$ | $44C^2$ | 25.78 | — |
| D=3 block_head (no corr_ffn) | $36C^2$ | 27.32 | 28.46 |
| Roformer $N{=}3$ | $36C^2$ | 27.19 | — |

Table A5: Adding the current token embedding to the correction input lowers parallel PPL relative to the token-blind correction at both tested depths. Concatenation obtains the lowest PPL. Models use $C = 446$ on Wikipedia.

| $D$ | Variant | Linear MACs | PPL | Seq $K{=}1$ | $L$ |
|---|---|---|---|---|---|
| 2 | corr_ffn (token-blind) | $32C^2$ | 26.68 | 26.72 | 0.74 |
| | corr_ffn_add | $32C^2$ | **26.09** | 26.48 | 0.54 |
| | corr_ffn_concat | $36C^2$ | **25.48** | 25.82 | 0.54 |
| 3 | corr_ffn (token-blind) | $44C^2$ | 23.98 | 23.96 | — |
| | corr_ffn_add | $44C^2$ | **23.79** | 24.12 | 0.55 |
| | corr_ffn_concat | $48C^2$ | **23.41** | 23.73 | 0.74 |

Table A6: In this sweep, context-ready has higher PPL at $C = 50$ and $C = 74$, and lower PPL at $C = 446$ and $C = 768$. Width and the correction FFN's share of the model both vary.

| $C$ | Comparison | Context-Ready | Baseline | $\Delta$ |
|---|---|---|---|---|
| 50 | D=3 vs. Roformer-hFFN $N{=}3$ | 84.3 | 83.0 | +1.3 |
| 74 | D=3 vs. Roformer-hFFN $N{=}3$ | 62.1 | 61.4 | +0.7 |
| 446 | D=3 vs. Roformer $N{=}4$ | 23.79 | 24.85 | $-1.06$ |
| 768 | D=3 vs. Roformer $N{=}4$ | 18.66 | 20.05 | $-1.39$ |

Context-ready has higher PPL at $C = 50$ and $C = 74$ and lower PPL at $C = 446$ and $C = 768$. The sweep compares the complete architectures as the correction FFN's share of the model changes with width. All main-body comparisons use $C \geq 256$.

Table A7: Setting $k_{\min} = 2$ lowers the measured correction-change ratio and the parallel $K{=}1$ PPL, while leaving the $K{=}10$ PPL similar. Models use $C = 50$, $D = 1$, and $K = 10$.

| Metric | $k_{\min} = 2$ | No $k_{\min}$ |
|---|---|---|
| Val PPL ($K = 10$) | 84.32 | **84.16** |
| Seq $K = 1$ | 84.61 | 84.19 |
| Parallel $K = 1$ | **118.35** | 130.95 |
| Empirical $L$ | **0.72** | 0.94 |

**Correction-change ratio.** With $k_{\min} = 2$, the measured ratio $\hat{L} \in [0.50, 0.72]$, where $\hat{L} = \max_t \|c_{K,t} - c_{K-1,t}\| / \|c_{K-1,t} - c_{K-2,t}\|$. Without $k_{\min}$, $\hat{L} \in [0.88, 1.20]$. This ratio is measured along the reported trajectories and is not the global Lipschitz constant $L$ in Theorem 3.

### C.3 $K = 10$ Training Details

All $K{=}10$ experiments: block size 1024, lr = 2e-4, softmax attention, $n_{\text{head}} = 16$, OWT.

Table A8: For $D$=1, fixed $K$=10 reaches 34.40 PPL at 100K iterations and 33.63 at 110K. Random-depth training reaches 33.63 at 135K, and fine-tuning reaches 31.35 at 215K total iterations. Runs use block size 1024 on OWT. Fine-tuning starts from an $N$=1 model pretrained for 85K iterations.

| **Strategy** | **PPL at 100K** | **Best PPL** | **Notes** |
|---|---|---|---|
| Roformer $N$=6 (baseline) | 35.37 | — | |
| $K$=10, fixed depth | **34.40** | 33.63 (110K) | Lower PPL at 65K–100K |
| $K$=10, $k_{\min}$=2 | 36.28 | 33.63 (135K) | +1.88 penalty at 100K |
| $K$=10, fine-tuned | 46.66* | **31.35** (215K) | Best final PPL |

*At 100K total (15K fine-tune); fine-tune reaches 35.92 at 150K total.

Table A9: Training trajectories for the fixed-$K$=10, $D$=1 model and the $N$=6 baseline. Both use batch size 16 and block size 1024.

| **Iter** | $N$=6 | $D$=1 $K$=10 | **Gap** |
|---|---|---|---|
| 40K | 43.24 | 43.83 | +0.59 |
| 60K | 39.21 | 39.27 | +0.06 |
| 65K | 38.55 | 38.49 | −0.06 |
| 80K | 36.99 | 36.38 | −0.61 |
| 100K | 35.37 | 34.40 | −0.97 |

### C.4 Sequential $K$=1 Validation

Full depth progression for all configurations. The agreement between sequential $K$=1 and parallel $K$=10 is a numerical check of the implementation against Theorem 2.

Table A10: For $D$=1, the first few block evaluations produce most of the $K$=5 improvement, and parallel $K$=10 and sequential evaluation agree within 0.02 PPL. Models use $C = 2048$ on OWT.

| **Block size** | **Iters** | **Par.** $K$=1 | **Par.** $K$=2 | **Par.** $K$=3 | **Par.** $K$=5 | **Par.** $K$=10 | **Seq.** $K$=1 |
|---|---|---|---|---|---|---|---|
| 256 | 100K | 84.44 | 43.77 | 40.52 | 39.95 | 40.02 | 40.03 |
| 256 | 400K | 70.80 | 36.21 | 33.23 | 32.70 | 32.79 | 32.80 |
| 512 | 100K | 81.02 | 38.25 | 35.01 | 34.35 | 34.41 | 34.43 |
| 512 | 200K | 73.22 | 34.33 | 31.24 | 30.61 | 30.68 | 30.69 |
| 1024 | 200K | 84.13 | 34.42 | 30.39 | 29.51 | 29.43 | **29.43** |

Table A11: As $D$ increases, fewer block evaluations approach the $K$=5 PPL. For $D \geq 8$, $K$=3 is within 0.12 PPL of $K$=5; parallel $K$=10 and sequential evaluation agree within 0.01 PPL at every tested depth.

| $D$ | **Par.** $K$=1 | **Par.** $K$=2 | **Par.** $K$=3 | **Par.** $K$=5 | **Par.** $K$=10 | **Seq.** $K$=1 |
|---|---|---|---|---|---|---|
| 5 | 58.17 | 40.18 | 38.80 | 38.60 | 38.61 | 38.62 |
| 8 | 51.60 | 40.12 | 39.22 | 39.10 | 39.10 | 39.10 |
| 12 | 38.33 | 32.58 | 32.30 | 32.29 | 32.29 | 32.29 |
| 23 | 32.80 | 29.03 | 28.89 | 28.88 | 28.88 | 28.88 |

At higher $D$, fewer block evaluations reach the $K$=5 PPL: $D$=12 and $D$=23 are within 0.01 PPL at $K$=3, and $D$=8 is within 0.12 PPL. The ratio between parallel $K$=1 PPL and the $K$=5 PPL decreases from 2.85× at $D$=1 to 1.14× at $D$=23.

### C.5 Block Size Scaling

Table A12 reports the $D{=}1$ and $N{=}6$ comparison at three block sizes. At $K{=}5$, the observed gap narrows from $+1.19$ at block size 256 to $+0.31$ at 1024. At $K{=}10$ and block size 1024, the $D{=}1$ run obtains lower PPL than the $N{=}6$ run (Section 5.5).

Table A12: At $K{=}5$, the PPL gap between $D{=}1$ and $N{=}6$ narrows as block size increases in this sweep. Models are evaluated at 200K iterations on OWT.

| **Block size** | $N{=}6$ **PPL** | $D{=}1$ **PPL** | **Gap** |
|---|---|---|---|
| 256 | 34.15 | 35.34 | +1.19 |
| 512 | 30.11 | 30.57 | +0.46 |
| 1024 | 29.22 | 29.53 | +0.31 |

### C.6 Token-Matched Training Curves

We compare $D{=}x$ context-ready against $N{=}x$ standard transformers at the same embedding dimension ($C = 1024$, block size 64) and token budget. In each run, context-ready PPL becomes lower after the reported crossover point.

Table A13: In these token-matched runs, context-ready obtains lower final PPL at every tested depth after the reported crossover point, with the largest observed margin at $D = 1$. Models use $C = 1024$, block size 64, and 1,126M training tokens on OWT.

| **Comparison** | $N{=}x$ **PPL** | $D{=}x$ **PPL** | **Gap** | **Crossover** |
|---|---|---|---|---|
| $D{=}1$ vs. $N{=}1$ | 114.8 | 80.4 | $-34.4$ | $\sim$424M |
| $D{=}2$ vs. $N{=}2$ | 73.7 | 66.1 | $-7.6$ | $\sim$565M |
| $D{=}3$ vs. $N{=}3$ | 62.4 | 59.4 | $-3.0$ | $\sim$835M |
| $D{=}6$ vs. $N{=}6$ | 53.0 | 52.2 | $-0.7$ | $\sim$1,032M |

### C.7 Fine-Tuning Details

A compatible pretrained $N$-layer decoder-only transformer can be converted to a $D{=}N$ context-ready model by adding a zero-initialized correction FFN and fine-tuning. At conversion, the correction is zero, so the model computes the same function as the original transformer. At $D{=}12$, fine-tuning causes a transient $+0.11$ PPL increase before recovering; at $D{=}24$, there is no transient increase.

Table A14: All three converted checkpoints obtain lower final PPL than their pre-conversion baselines. In the controlled $N{=}12$, $C{=}1408$ run, the converted model also obtains lower PPL than continued roformer training. Models use block size 256 on OWT.

| **Conversion** | **Baseline** | **Fine-tuned** | **Cont. baseline** | **$\Delta$ vs. cont.** | **FT iters** |
|---|---|---|---|---|---|
| $N{=}12\ C{=}1408 \to D{=}12$ | 29.92 | 26.14 | 27.20 | $-1.06$ | 200K |
| $N{=}24\ C{=}1024 \to D{=}24$ | 29.42 | 28.99 | — | $-0.43^*$ | 18K |
| $N{=}12\ C{=}1024 \to D{=}12$ | 33.41 | 32.21 | — | $-1.20^*$ | 50K |

$^*$Gain vs. pre-conversion checkpoint; continued-training control not available.

### C.8 Wikipedia Results

We also evaluate on English Wikipedia (BPE 16k, context 256, 100K iterations).

Table A15: Context-ready obtains lower PPL in all three reported comparisons at matched linear-map cost on Wikipedia. Counts include block and correction costs and omit the prediction head, which is identical within each width group.

| | Model | Linear MACs/tok | Val PPL | Δ |
|---|---|---|---|---|
| $C$=768 | D=5 concat | 42.5M | **16.69** | |
| | Roformer $N$=6 | 42.5M | 17.95 | −1.26 |
| $C$=446 | D=6 add | 15.9M | **20.40** | |
| | Roformer-hFFN $N$=6 | 15.9M | 21.44 | −1.04 |
| | D=2 concat | 7.2M | **25.48** | |
| | Roformer $N$=3 | 7.2M | 27.19 | −1.71 |

### C.9 Inference Timing

Autoregressive generation speed measured on a single A100 GPU over 10,000 tokens with KV caching, batch size 1 (single-sequence generation).

Table A16: The tested context-ready configurations generate 2.6× and 1.7× faster than the listed roformers in these cross-width comparisons. Measurements use one A100, 10K generated tokens, batch size 1, and KV caching.

| Model | Params | tok/s | ms/tok | Speedup |
|---|---|---|---|---|
| $D$=1 $C$=2048 | 215M | 919 | 1.09 | 2.6× |
| Roformer $N$=6 $C$=1088 | 155M | 351 | 2.85 | |
| $D$=5 $C$=1120 | 157M | 349 | 2.86 | 1.7× |
| Roformer $N$=12 $C$=768 | 134M | 201 | 4.96 | |

Measured per-token latency grows by less than 4% from $T$=100 to $T$=10,000 in both comparisons, even though attention work per new token is $O(T)$. These measurements compare the complete deployed configurations. Fewer layers reduce total KV cache memory ($C \times D$ per token): $D$=1 $C$=2048 uses 3.2× less cache than $N$=6 $C$=1088, and $D$=5 $C$=1120 uses 1.6× less than $N$=12 $C$=768.