INTERNALIZING CHAIN-OF-THOUGHT INTO LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

The expressive power of transformers concerns both what they can compute and the resources required, including depth, precision, context length, and generated tokens. We organize existing constructions by where they store intermediate states. *In-layer* constructions keep them in hidden activations and produce the answer in one generation step, whereas *in-CoT* constructions record the computation in generated tokens and keep the workspace in the context window. We construct a family of constant-bit-size looped transformers that interpolates between these two classes. For a prescribed loop count $L = L(n)$ and a fixed integer $k \geq 1$ satisfying $s(n)^{1/k} = \Omega(L)$, our transformer simulates a time-$t(n)$, space-$s(n)$ Turing machine using a context window of $O(s(n)/L)$ and $O(t(n)s(n)^{1/k}/L)$ generated tokens, reducing both resources by a factor of L relative to the corresponding unlooped queue-machine simulation. The construction also holds with the finite-precision softmax kernel specified in this paper.

1 INTRODUCTION

The expressive power of transformers asks what they can compute and what resources they require, including depth, precision, context length, attention heads, and generated tokens. Formal-language theory provides one route to these questions by studying which languages transformers can recognize. RASP models a transformer as a human-readable program (Weiss et al., 2021). Subsequent work characterizes the languages recognized under different architectural restrictions (Angluin et al., 2024; Yang & Chiang, 2024; Zhou et al., 2024; Strobl et al., 2025; Huang et al., 2025). These constructions keep intermediate states in hidden activations and produce the answer in one generation step; we call them *in-layer* constructions. They expose how language structure and algorithmic parallelism determine the resources required for one-step generation.

Chain-of-thought prompting instead uses autoregressively generated tokens for intermediate computation (Wei et al., 2022), extending the model's expressive power through recurrence without increasing its bit-size. Pérez et al. (2021) proved that transformers with autoregressive generation are Turing-complete. We call constructions of this form *in-CoT*.

The context window is then a computational resource. Early Turing-completeness constructions use growing precision and full-history context that can scale with running time (Pérez et al., 2021; Bhattamishra et al., 2020; Merrill & Sabharwal, 2024). Standard full-history CoT with a polynomial-time next-token map and polynomial context cannot decide all of PSPACE without further computational conjecture. Sliding-window and summarized-CoT simulations reduce visible context to $O(s(n))$ (Schuurmans et al., 2024; Yang et al., 2025; Brösamle & Eckstein, 2026). Li & Wang (2025a) achieve Turing completeness with a constant-bit-size transformer attending only to fixed relative offsets.

Looped transformers provide another form of recurrence by repeatedly applying a shared block to hidden states in-layer (Giannou et al., 2023; Yang et al., 2024; Geiping et al., 2025). They can perform multiple such iterations before emitting a token, carrying intermediate results in hidden representations rather than an explicit CoT trace. This internal reasoning is a form of *latent thought* (Saunshi et al., 2025; Xu & Sato, 2026). These connections motivate studying the tradeoff between latent loop iterations and explicit CoT steps.

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.

Our contribution. We construct a family of constant-bit-size looped transformers that interpolates between latent thought and chain of thought. Our looped transformer applies one shared decoder block L times per generated token. Relative to the queue-machine simulations of Li & Wang (2025a;b), grouping L simulated transitions into each generated token reduces both the context window and the CoT length by a factor of L, provided that the minimum queue length is $\Omega(L)$. The same construction holds with softmax attention under the finite-precision kernel of Section 5. More specifically,

Theorem (informal)**.** For any $(t(n), s(n))$ time-space bounded Turing machine, any fixed integer $k \geq 1$, and any loop count $L = L(n) = O(s(n)^{1/k})$, there is a constant-bit-size looped transformer that simulates the machine with context window $O(s(n)/L)$ in

$$T_{\text{cot}} = O\left(\frac{t(n)s(n)^{1/k}}{L}\right)$$

CoT steps. At $L = 1$, the bounds recover (Li & Wang, 2025b, Thm 1).

Proof idea. The basic idea is simple: we fold the computation that would normally be externalized as chain-of-thought tokens into the intermediate activations of the looped transformer. A loop iteration performs one computational step and keeps its result in the latent state, so that L loop iterations can replace L explicitly generated CoT tokens. The main challenge is that internalizing a sequence of computations changes how intermediate states can be accessed. In ordinary CoT, all tokens within the context window are simultaneously available to every attention layer. In a transformer, however, computation is organized across layers, and an attention operation at a given layer can only access the cached activations produced at that same layer. Thus, folding an existing CoT simulation into the loop is not merely a matter of replacing tokens by hidden states: the information needed at each computational step must also be aligned with the layer at which that step is executed. We do not know how to obtain such an alignment for many existing Turing-completeness constructions. Our key technical observation is that Li–Wang's queue-machine simulation admits a suitable reorganization: the queue representation can be modified so that, at every recurrent step, all information required by the simulated transition is stored at the corresponding layer of the KV cache. This layer-aligned representation allows one shared block to execute L consecutive machine steps within a single generated token, yielding the desired tradeoff between recurrent computation, visible context, and explicit CoT.

Potential Messages for real LLMs

Beyond its expressivity implications, our construction also suggests several messages for practical reasoning models.

- First, it shows that looped transformers have sufficient expressive capacity to internalize a substantial portion of chain-of-thought computation into recurrent latent state: increasing the loop count exchanges explicitly generated reasoning tokens for computation performed within the loop, while preserving general-purpose computational expressivity. This makes looped architectures a natural candidate for separating reasoning computation from visible token generation.
- Second, the construction suggests that the number of generated CoT tokens need not faithfully reflect the amount of serial computation performed by a model. In our simulation, one generated token summarizes L consecutive computational steps, with the intermediate states retained in the recurrent/KV state rather than explicitly externalized as tokens. Thus, visible CoT length and underlying reasoning computation can in principle be substantially decoupled.
- Third, this decoupling also exposes a potential learning challenge at large L: the explicit reasoning trace becomes increasingly strided, since only every L-th computational state is exposed as a token, while intermediate computation remains latent. Such a trajectory is not directly supervised by the standard next-token prediction objective, which may make this form of computation difficult to acquire through pretraining alone.
- This perspective is qualitatively consistent with recent observations that reinforcement-learning-trained reasoning models can develop irregular or substantially more concise reasoning traces: for example, DeepSeek-R1-Zero exhibits strong reasoning behavior despite

poor readability and language mixing, while subsequent work has shown that reinforcement learning can explicitly induce much shorter reasoning traces without sacrificing task performance. Such observations are consistent with the possibility that outcome-driven RL places weaker constraints on the textual form of intermediate computation than language-model pretraining, allowing reasoning computation to become less tightly coupled to a smooth, natural-language-like CoT.

- Finally, our construction points toward architectural and training strategies for long-reasoning models in which latent computation, persistent state, and visible reasoning tokens are separately controlledfor example, by treating the loop count as an explicit latent-compute budget and exposing or supervising only selected boundary states rather than requiring every internal computation step to be verbalized.

Section 2 surveys the two families, including related works on chain-of-thought and latent thought. Section 3 fixes the model. Section 4 gives the hardmax construction, with initialization (Section 4.3) stated explicitly. Section 5 lifts it to softmax at constant bit-size. Appendix A constructs congruent queue lengths; Appendix B records the transformer components.

2 Related Work

2.1 In-layer constructions

Formal-language theory provides one route to in-layer expressivity by studying which languages transformers recognize. RASP models a transformer as a human-readable program (Weiss et al., 2021), and Tracr compiles such programs into transformer weights (Lindner et al., 2023); subsequent work characterizes the languages recognized under different architectural restrictions (Angluin et al., 2024; Yang & Chiang, 2024; Zhou et al., 2024; Strobl et al., 2025; Huang et al., 2025). On the circuit side, Merrill & Sabharwal (2023) established that constant-depth, log-precision transformers without CoT are limited to TC^0-style shallow circuits. These constructions are one-pass: each layer is applied once, so the number of sequential computation steps equals the number of layers. Looped transformers instead re-apply a shared block to the hidden state, decoupling serial computation from the layer count; intermediate results are carried in the latent state across iterations. Giannou et al. (2023); Yang et al. (2024); Geiping et al. (2025) loop a constant-size block directly, and Merrill & Sabharwal (2025) show that looped transformers of logarithmic depth recognize regular languages and solve graph connectivity without CoT — two problems inexpressible at fixed depth under $\mathsf{TC}^0 \neq \mathsf{NC}^1$. Saunshi et al. (2025) show that loop iterations can simulate CoT steps one-for-one, and Xu & Sato (2026) compare the two modes formally: sufficiently many loops exceed the power of CoT with the same iteration budget, whereas stochastic CoT supports approximate counting that deterministic latent thought cannot. Both analyses let the embedding dimension grow with the input length. Feng et al. (2023) connect CoT length to circuit depth: polynomially long CoT lifts transformers from shallow-circuit problems to all of P.

2.2 In-CoT constructions

Pérez et al. (2021) proved that hard-attention transformers are Turing-complete by directly simulating TMs; a bounded-precision variant captures $\mathrm{Time}(t(n))$ with $\log t(n)$ bits. Bhattamishra et al. (2020) used unbounded precision; Merrill & Sabharwal (2024) $O(\log(n + t(n)))$ precision; Li et al. (2024) constant precision and $O(\log n)$ dimension; Qiu et al. (2025) prompting-based completeness at $O(\log(n + t(n)))$ precision. Jiang et al. (2026) establish softmax Turing completeness via counter-machine simulation with full-history attention and no fixed numerical-precision bound.

Repair 1: context management. Yang et al. (2025) (PENCIL) recursively erase intermediate CoT, so the window scales with space rather than time: maximal context $O(s(n))$, and with polynomial context PENCIL solves all of PSPACE while standard CoT solves at most P (their Corollary 5.2). Brösamle & Eckstein (2026) simulate TMs with low-precision softmax and summarized CoT, keeping every segment at $O(s(n))$ at the price of logarithmically growing depth, width, and precision. Cui et al. (2026) show that the management policy determines the computable class: fixed summarization-style systems can be simulated by TMs in constant space, and fixed appending-style systems in linear space.

Repair 2: sliding-window attention. Schuurmans et al. (2024) gave the earliest bounded-window universality (Lag system; context as a circular queue). Li & Wang (2025a) prove constant-bit-size transformers Turing complete with an $O(s(n))$ window via a Post-machine simulation and characterize $\mathsf{WINDOW}[s(n)] = \mathsf{SPACE}[s(n)]$. Li et al. (2026) show Turing completeness without positional encodings using a window of length at most $s(n) + 1$ via a parity-HIST abstraction. Li & Wang (2025b) achieve $O(s(n)^{1/k})$ CoT tokens per TM step with window $s(n) + 1$ by increasing the head–layer product K.

Source	Precision	Dim.	Depth	Window	CoT per TM-step
Pérez et al. 2021	$O(\log t)$	$O(1)$	$O(1)$	$n + t$	1
Bhattamishra et al. 2020	unbounded	$O(1)$	1	$n + t$	1
Merrill & Sabharwal 2024	$O(\log t)$	$O(1)$	$O(1)$	$n + t$	1
Li et al. 2024	$O(1)$	$O(\log t)^{\ddagger}$	2	$O(t \log t)$	$O(\log t)$ amort.
Qiu et al. 2025	$O(\log t)$	$O(1)$	$O(1)$	$O(t \log t)$	$O(\log t)$ amort.
Li & Wang 2025a	$O(1)$	$O(1)$	1	s	s
Li & Wang 2025b	$O(1)$	$O(1)$	—†	$s{+}1$	$O(s^{1/k})$
Jiang et al. 2026$^{\diamond}$	unbounded	$O(1)$	$O(1)$	$n + 2^{O(t)}$	1 per counter step
Brösamle & Eckstein 2026$^{\diamond}$	$O(\log \log s)$	$O(\log s)$	$O(\log s)$	$O(s)$ (SCoT)	$O(1)$ amort.
This work$^{\diamond}$	$O(1)$	$O(1)$	L	$O(s/L)$	$O(s^{1/k}/L)$

$^{\diamond}$ marks constructions using softmax attention (this work via the finite-precision kernel of §5).
† Li & Wang 2025b report only the head–layer product $K = H{\cdot}L$; their depth is left unspecified.
‡ Li et al. (2024) state $O(\log n)$ embedding dimension under the assumption that $T(n)$ is polynomial, so $\lceil \log_2(T(n) + n) \rceil = O(\log n)$.

Table 1: Comparing transformers simulating (t, s)-bounded Turing machines. Our depth is the number L of loop executions of one shared decoder block. This work prescribes L and chooses k subject to $s(n)^{1/k} = \Omega(L)$; Li & Wang 2025b is the endpoint $L{=}1$.

3 Preliminaries

3.1 Models of computation

Multi-tape Turing machine. A k-tape Turing machine (TM) is a tuple $\langle \Sigma, Q, \delta \rangle$ where

- Σ is a finite tape alphabet, including 0, 1, and a blank symbol $\#$.
- Q is a finite set of states, including a start state q_{start} and halting states $q_{\text{acc}}, q_{\text{rej}}$.
- $\delta : Q \times \Sigma^k \to Q \times \Sigma^{k-1} \times (\{\text{Left}, \text{Stay}, \text{Right}\})^k$ is a transition function.

The first tape is a read-only input tape, initially holding a finite $\{0, 1\}$-string of input x; the remaining $k{-}1$ tapes start empty. A TM is $(t(n), s(n))$ time-space bounded if, on every input of length n, it halts within $t(n)$ steps and visits at most $s(n)$ distinct cells across the k tapes.

Multi-queue Turing machine. A k-queue Turing machine is a tuple $\langle \Sigma, Q, \delta \rangle$, the multi-queue generalization of the Post machine (Post, 1936). The difference from the TM lies in the access mode: at every step the machine appends one symbol at each rear, while at each front the transition function decides whether the head stays or moves one cell to the right.

- Σ is a finite alphabet, containing 0, 1, and the blank $\#$.
- Q is a finite state set, containing the start state q_{start} and the halting states $q_{\text{acc}}, q_{\text{rej}}$.
- $\delta : Q \times \Sigma^k \to Q \times \Sigma^k \times (\{\text{Stay}, \text{Right}\})^k$ is the transition function, specifying the k appended symbols and the k front-head moves.

We use an adapted version, called the **synchronous multi-queue TM** (Li & Wang, 2025b, §2.2):

1. Initially, the input queue contains the input string, and the other $k-1$ queues contain only blank symbols.
2. All queues pop and push synchronously: every front head moves right by one cell (its symbol is popped) and one symbol is appended at every rear. Consequently the $\{\mathrm{Stay}, \mathrm{Right}\}^k$ component is always Right, thus simplifying the transition function to $\delta : Q \times \Sigma^k \to Q \times \Sigma^k$.

The space complexity of a queue machine is the maximum number of symbols held in its queues during the computation, over all inputs of length n. In the rest of the paper $s(n)$ refers only to the simulated TM's space bound; the queue machine uses $O(s(n))$ space.

Without loss of generality, queue 0 is the (longest) input queue and every halting state is absorbing: for $q \in \{q_{\mathrm{acc}}, q_{\mathrm{rej}}\}$ and every K-tuple $\sigma \in \Sigma^K$, set $\delta(q, \sigma) = (q, \#, \ldots, \#)$.

Definition 1 (K-queue execution log)**.** Let $M = (\Sigma, Q, \delta)$ be a synchronous K-queue machine with fixed queue lengths $\ell_0, \ldots, \ell_{K-1}$, on input $x = x_0 \cdots x_{n-1}$ of length $n \le \ell_0$, and let $s_{\max} = \max_r \ell_r$. Its execution log is $(\sigma_k, q_k)_{k \ge 0}$, where $\sigma_k = (\sigma_k^{(0)}, \ldots, \sigma_k^{(K-1)})$. The entries with $k \le s_{\max} - 1$ are given by definition: the input feed, for $0 \le k \le n-1$,

$$(\sigma_k,\, q_k) = (x_k,\, \#^{K-1},\, q_{\mathrm{start}}),$$

and the padding, for $n \le k \le s_{\max} - 1$,

$$(\sigma_k,\, q_k) = (\#^K,\, q_{\mathrm{start}}).$$

For $k \ge s_{\max} - 1$,

$$(\sigma_{k+1},\, q_{k+1}) = \delta\big(q_k,\, \sigma_{k-\ell_0+1}^{(0)},\, \ldots,\, \sigma_{k-\ell_{K-1}+1}^{(K-1)}\big),$$

every front index $k - \ell_r + 1 \ge s_{\max} - \ell_r \ge 0$ being given or previously produced; the absorbing convention makes the log infinite. The remaining run length after the cache is filled is

$$\tau(n) = \max\big(1,\, \min\{k \mid q_k \in \{q_{\mathrm{acc}}, q_{\mathrm{rej}}\}\} - (s_{\max} - 1)\big).$$

Fact 2 (Queue Turing machines are Turing-complete (Li & Wang, 2025b, Thm. 2, Step I))**.** For every fixed integer $k \ge 1$, a $t(n)$-time, $s(n)$-space k'-tape Turing machine converts to a synchronous $K = 6kk'$-queue machine running in $O(t(n)s(n)^{1/k})$ steps and $O(k's(n))$ total queue space. Each tape becomes two stacks and each stack k levels of queues. Writing $B = \lceil s(n)^{1/k} \rceil$, level i ($1 \le i \le k$) uses two content-half queues of length B^i and one buffer of length $2B^i$; consecutive half lengths divide one another, and the cost analysis is in terms of these lengths (Li & Wang, 2025b, §3).

3.2 Sliding-window transformers with KV cache

Decoder-only transformer. Let $\mathcal{V}$ be the vocabulary. A decoder-only transformer is a parameterized next-token map $\mathcal{V}^* \to \mathcal{V}$. The forward pass for position i sets $h_i^{(0)} = \mathrm{emb}(v_i)$ and, for layers $l = 0, \ldots, L-1$,

$$h_i^{(l+0.5)} = W_O^{(l)} \cdot \mathrm{concat}_h\Big(\sum_j s_{h,i,j}^{(l)} \cdot W_v^{(l,h)}\, h_j^{(l)}\Big), \qquad \tilde{h}_i^{(l)} = h_i^{(l+0.5)} + b^{(l)} + h_i^{(l)},$$

$$h_i^{(l+1)} = \mathrm{FF}^{(l)}\big(\tilde{h}_i^{(l)}\big) + \tilde{h}_i^{(l)},$$

where the inner sums range over the context window and the concatenation runs over the H heads. The learned parameters comprise the embedding and output head, query and key matrices $W_q^{(l,h)}, W_k^{(l,h)} \in \mathbb{R}^{d_k \times d}$, value matrices $W_v^{(l,h)} \in \mathbb{R}^{d_v \times d}$, output matrices $W_O^{(l)} \in \mathbb{R}^{d \times H d_v}$, biases $b^{(l)}$, and feed-forward networks $\mathrm{FF}^{(l)}$. We specify the relative positional function $\mathrm{pos}^{(l)} : \mathbb{N} \to \mathbb{R}^{d_k}$ separately. The weights $s_{h,i,\cdot}^{(l)}$ are the hardmax distribution over scores $\langle W_q^{(l,h)} h_i^{(l)},\, W_k^{(l,h)} h_j^{(l)} + \mathrm{pos}^{(l)}(i-j)\rangle$. The next token is $v_{i+1} = \mathrm{argmax}(W_{\mathrm{out}} h_i^{(L)})$.

In the **sliding-window** setting with window w, position i attends to $\{i-w+1, \ldots, i\} \cap \mathbb{N}$. The **KV cache** is indexed by (position, layer): $W_k^{(l,h)} h_j^{(l)}$ and $W_v^{(l,h)} h_j^{(l)}$ are written and fixed when position j completes layer l.

Remark. A transformer is often viewed as a stateless map from its visible context to the next token. With sliding-window attention and a persistent KV cache, retained states can depend on tokens that have already left that window. Each layer can extend the dependence span by $w-1$ positions, so an L-layer pass can have a receptive field of $L(w-1)+1$ token positions. This dependence span is distinct from both the visible window and the amount of cached state.

3.3 LOOPED TRANSFORMERS

Looped transformers re-apply a fixed unit instead of stacking independently parameterized layers (Giannou et al., 2023; Yang et al., 2024; Geiping et al., 2025). We adopt a *weight-tying* version (Dehghani et al., 2019; Lan et al., 2020): the unit acts L times on the current position's residual stream per generated token, with a separate per-iteration KV cache carrying state across positions.

Definition 3 (looped transformer)**.** A *looped transformer* is a decoder-only transformer consisting of

- a **loop unit** u_φ: a stack of $m \geq 1$ decoder layers specified by the shared weights φ;
- the embedding emb and the output head W_{out}, applied outside the loop;

together with a **loop count** $L \in \mathbb{N}$. Given L, the forward pass computes

$$h_i^{(0)} = \mathrm{emb}(v_i), \qquad h_i^{(l+1)} = u_\varphi\big(h_i^{(l)}; \mathcal{C}_i^{(l)}\big) \quad (0 \leq l < L),$$
$$v_{i+1} = \mathrm{argmax}\big(W_{\mathrm{out}}\, h_i^{(L)}\big),$$

the l-th application of u_φ being iteration l (layer and iteration are used interchangeably). Here $\mathcal{C}_i^{(l)}$ denotes the layer-l KV entries at retained positions before i; the current position's key and value are computed from $h_i^{(l)}$. Unrolling the loop yields an mL-layer transformer in which each iteration applies the same m layers. Shared parameters are independent of L.

A looped transformer is *constant-bit-size* if the number of distinct learned parameters, their bit precision, and the bit precision of scalar intermediate values are all $O(1)$, independently of input length, loop count, and generation length.

4 THE HARDMAX CONSTRUCTION

We first construct the simulation with hardmax attention; Section 5 realizes the same selection exactly under a finite-precision softmax kernel. Throughout this section, queue r has length $\ell_r \equiv 1 \pmod{L}$ (the general case is Appendix A). Set $w = \max_r w_r$ and $s_{\max} = \max_r \ell_r$. Each forward pass executes L consecutive machine steps, with head r reading queue r's front at distance $w_r - 1$.

4.1 TOKENS AND RESIDUAL LAYOUT

At nonterminal position i, the token is (q_{Li}, σ_{Li}), where $\sigma_{Li} = (\sigma_{Li}^{(0)}, \ldots, \sigma_{Li}^{(K-1)}) \in \Sigma^K$ is the tuple appended at log index Li.

The embedding uses the layout specified in Appendix B.1:

$$h_i^{(0)} = \big(1,\ e_{q_{Li}},\ \mathrm{onehot}(\sigma_{Li}^{(0)}), \ldots, \mathrm{onehot}(\sigma_{Li}^{(K-1)}),\ 0^{|\Sigma|K}\big).$$

The blocks store respectively a constant, the current state, the appended tuple, and K scratch registers for retrieved fronts, giving $d = 2|\Sigma|K + |Q| + 1 = \Theta(K)$. At iteration l, attention writes the fronts and the feed-forward network applies δ, leaving the residual equal to the embedding of $(q_{Li+l+1}, \sigma_{Li+l+1})$. After iteration $L-1$, the output head emits $(q_{L(i+1)}, \sigma_{L(i+1)})$, or ACCEPT/REJECT if the state is halting, and the generation stops.

4.2 POSITIONAL ENCODING AND THE ATTENTION HEADS

The relative positional encoding is the simplified form of Li & Wang (2025b, Eq. (2)): writing e_r for the r-th standard basis vector of $\mathbb{R}^{d_k}$, $d_k = K$,

$$\mathrm{pos}(i-j) \;=\; \Delta \sum_{r:\, i-j=w_r-1} e_r,$$

whose r-th coordinate is Δ exactly at distance $i - j = w_r - 1$. The query of head r is the constant e_r and the key is set to be 0. The positional encoding enters the key side of the scores (§3.2). Thus

$$\mathrm{score}_r(i, j) = \langle e_r,\ \mathrm{pos}(i - j)\rangle = \Delta[\, i - j = w_r - 1\,].$$

For every pass $i \geq w - 1$, $j^* = i - w_r + 1$ is the unique maximizer, so hardmax selects queue r's designated position. Equal-length queues remain separate via distinct heads and value projections.

Value entries and transition. Let $V_r(j, l) = W_v^{(r)} h_j^{(l)} \in \mathbb{R}^{d_v}$, $d_v = |\Sigma|$, be head r's cached value at position j and iteration l, and let $V(j, l)$ denote the represented K-tuple. Each $W_v^{(r)}$ projects symbol block r; W_O writes the selected values into the front blocks, and the feed-forward network applies δ. All L iterations share the same positional encoding, value projections, and δ lookup, so the construction satisfies Definition 3.

4.3 The loop invariant

Initialization. Generation begins at pass $w-1$ with a loaded cache: for $0 \leq i \leq w-2$ and $0 \leq l \leq L - 1$, set the cached value entry $V(i, l) = \sigma_{Li+l}$, and set the current token to $(q_{L(w-1)}, \sigma_{L(w-1)})$. Since $L(w - 1) = s_{\max} - 1$, the first simulated transition occurs at step $s_{\max} - 1$; the remaining run length is $\tau(n)$ (Definition 1).

Lemma 4 (loop invariant)**.** *For every position i and layer l, the value entry $V(i, l)$ encodes the K-tuple at log index $Li + l$: block r holds $\sigma^{(r)}_{Li+l}$.*

Proof. The initialization sets $V(i, l) = \sigma_{Li+l}$ for $0 \leq i \leq w - 2$ and all l, while the token at position $w - 1$ supplies $V(w - 1, 0) = \sigma_{L(w-1)}$.

Consider pass $t \geq w - 1$ and suppose the residual entering iteration l contains $(q_{Lt+l}, \sigma_{Lt+l})$. Head r attends to position $t - w_r + 1$ and retrieves

$$V(t - w_r{+}1,\ l)^{(r)} \ =\ \sigma^{(r)}_{L(t-w_r+1)+l} \ =\ \sigma^{(r)}_{Lt+l-\ell_r+1},$$

the front of queue r for step $Lt+l$, because $\ell_r - 1 = L(w_r - 1)$. The residual stream supplies q_{Lt+l}, and the MLP computes

$$(q_{Lt+l+1},\ \sigma_{Lt+l+1}) \ =\ \delta\big(q_{Lt+l},\ \sigma^{(0)}_{Lt+l-\ell_0+1}, \ldots, \sigma^{(K-1)}_{Lt+l-\ell_{K-1}+1}\big).$$

For $l < L - 1$, this result is cached as $V(t, l + 1)$; for $l = L - 1$, the output head emits $(q_{L(t+1)}, \sigma_{L(t+1)})$, whose embedding supplies $V(t + 1, 0)$. Induction over iterations and passes proves the claim. □

4.4 Proof of the main theorem

Theorem 5. *For any $(t(n), s(n))$ time-space bounded k'-tape Turing machine, any fixed integer $k \geq 1$, and any loop count $L = L(n) = O(s(n)^{1/k})$, there exists a constant-bit-size looped transformer that, with its input encoded in the initial KV cache as specified in Section 4.3, simulates the machine with visible context window $O(s(n)/L)$ in*

$$T_{\mathrm{cot}} = O\left(\frac{t(n)s(n)^{1/k}}{L}\right)$$

CoT steps.

We prove it in two steps: Lemma A.1 reduces the k'-tape Turing machine to a congruent queue machine; Proposition 6 shows that the construction of §§4.1–4.3 simulates that machine.

Proposition 6 (queue-level construction)**.** *Let $M = (\Sigma, Q, \delta)$ be a synchronous K-queue machine over a fixed alphabet Σ, with absorbing halting states and lengths $\ell_r = L(w_r - 1) + 1$. Under the initialization of §4.3, the construction of §§4.1–4.3 gives a decoder-only looped transformer with one shared block, K heads, window $w = \max_r w_r$, embedding dimension $2|\Sigma|K + |Q| + 1$, and constant bit-size. It emits the execution log at stride L and therefore uses*

$$T_{\mathrm{cot}} = \left\lceil \frac{\tau(n)}{L} \right\rceil$$

generated tokens, including at most $L - 1$ absorbing transitions in the final pass.

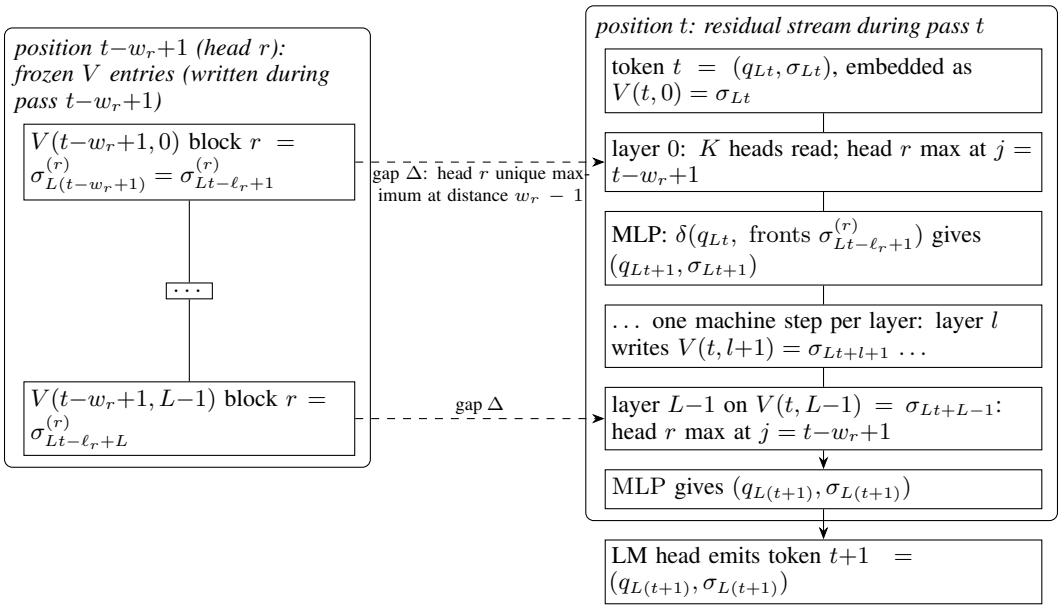

Figure 1: One pass at position t. At iteration l, head r retrieves queue r's front from $V(t-w_r+1, l)$, and the feed-forward network advances the machine by one step. The final write is emitted as token $t+1$.

Proof. The decoder block, K heads, window w, dimension $2|\Sigma|K + |Q| + 1$, and constant bit-size are those of §§4.1–4.2. By Lemma 4, at every pass $t \geq w - 1$ and every iteration l the cached values are the log entries σ_{Lt+l}, so head r retrieves the front of queue r at machine step $Lt + l$ and the feed-forward network applies δ. After L iterations the residual is therefore the embedding of $(q_{L(t+1)}, \sigma_{L(t+1)})$. Appendix B.6 emits this pair as token $t + 1$, or ACCEPT/REJECT if the state is halting. Generation thus starts from the initialization of §4.3 at index $L(w - 1) = s_{\max} - 1$ and follows the log at stride L for the remaining $\tau(n)$ steps, giving $T_{\text{cot}} = \lceil \tau(n)/L \rceil$ with at most $L - 1$ absorbing transitions in the final pass. □

Proof of Theorem 5. For the prescribed L, choose a fixed k satisfying $s(n)^{1/k} = \Omega(L)$. The congruent lengths supplied by Appendix A (Lemma A.1) give a step-equivalent $K = 8kk'$-queue simulation with run length $\tau(n) = O(t(n)s(n)^{1/k})$ and window $w = O(s(n)/L)$. Proposition 6 yields

$$T_{\text{cot}} = O\left(\frac{t(n)s(n)^{1/k}}{L}\right).$$

Multiplying by L gives the serial-work bound. The cache stores w positions at each of L iterations, so total KV state is $\Theta(Lw) = \Theta(s(n))$.

5 Softmax Simulates Hardmax at Constant Bit-Size

Section 4 uses hardmax attention. We now show that Theorem 5 holds under the max-first binary32 softmax kernel below, without increasing precision as the window or loop count grows.

Write $s_{i,j}$ for the attention score of candidate position j in the row of query position i. For a row with unique maximizer j^*, define the attention score gap

$$\Delta \;=\; s_{i,j^*} \;-\; \max_{j \neq j^*} s_{i,j},$$

the score separation of (Brösamle & Eckstein, 2026, Def. D.1).

The max-first kernel. Every attention score is a finite binary32 value (IEEE single precision; RN_{32} denotes round-to-nearest in binary32). Given a score row $(s_j)_j$ and value vectors $(v_j)_j$, the kernel computes the attention output as follows:

- (i) compute the row maximum $m = \max_j s_j$;
- (ii) form the shifted inputs $d_j = \mathrm{RN}_{32}(s_j - m)$;
- (iii) apply a fixed binary32 exponential primitive E_{32} with $E_{32}(0) = 1$ and $E_{32}(x) = +0$ for every $x \le -256$;
- (iv) accumulate the numerator $\sum_j E_{32}(d_j)\, v_j$ and the normalizer $\sum_j E_{32}(d_j)$ in binary32, dividing only after both sums;
- (v) cast the quotient once, at the output boundary, to a fixed format containing 0 and 1 exactly.

Since $256 > 150 \ln 2$, we have $e^{-256} < 2^{-150}$, the round-to-nearest midpoint between 0 and the least positive binary32 subnormal 2^{-149}, so a correctly rounded binary32 exponential returns $+0$ on every input at most -256. Flush-to-zero only strengthens the property. Any cutoff larger than $150 \ln 2$ in magnitude has the same property; $256 = 2^8$ is chosen so that the multiplier of Corollary 8 is a power of two.

Lemma 7 (softmax implements unique hardmax exactly)**.** *Let $(s_j)_j$ be a score row with unique maximizer j^* and score gap at least 256. Then the max-first kernel's computed attention output is exactly v_{j^*}, independently of the row length.*

Proof. By the kernel, the shifted inputs are $d_{j^*} = 0$ and $d_j \le -256$ for every $j \ne j^*$. The exponential row is exactly the one-hot indicator of j^*: $E_{32}(d_{j^*}) = 1$ and $E_{32}(d_j) = +0$ for every $j \ne j^*$. Numerator and denominator are therefore the exact sums v_{j^*} and 1, each partial sum containing only zeros and one nonzero term. Division and the single boundary cast preserve the result. □

Remark. (1) The lemma requires a binary32 exponential that returns $+0$ below a fixed threshold, without positive floor or logit soft-capping. (2) Subtracting the maximum converts overflow into underflow and thereby helps to simulate hard selection. We include it in our softmax kernel because it is standard in production implementations.[1] For a kernel that does not subtract the maximum, the positional encoding of §4.2 can set the winner's score to 0 and every other score to -256. (3) If the kernel includes a positive scale, canonically $1/\sqrt{d_k}$, instantiate $\Delta = \lceil 256\sqrt{d_k} \rceil$. Here $d_k = K$ is a machine constant, so Δ remains $O(1)$.

Corollary 8 (the main results hold under softmax)**.** *Set $\Delta = 256$ in §4.2. Then every score row has scores in $\{256, 0\}$, a unique maximizer at distance $w_r - 1$, and shifted losers exactly -256, all representable (binary16 parameters, binary32 internals, values in $\{0, 1\}$). By Lemma 7 every attention output equals the hardmax one-hot of §4.2 bit for bit, so Theorem 5 and Proposition 6 hold with softmax attention under the max-first kernel: same parameters up to the fixed multiplier, same resources, bit-size still $O(1)$ independent of n and L.*

The contrast with exact-real arithmetic is sharp. Brösamle & Eckstein (2026) preserve hard selection at the cost of a logarithmically growing factor plus a denoising layer per decoder layer, in order to separate the winner from the sum of the remaining scores; underflow of the finite-format exponential then makes the separation exact. When the transformer has constant bit-size and rounds at each step — as a production softmax kernel does — we can instead choose a constant that does not overflow and underflows after exponentiation, and thereby simulate hardmax at no extra cost.

AI USE STATEMENT

We used generative AI tools to improve the readability of the manuscript and to identify relevant literature. We also used the Paper Assistant Tool (PAT) to review the paper's technical content. After assessing its feedback, we corrected errors it identified and cited relevant papers it suggested. The authors reviewed the AI-assisted revisions and take responsibility for the final text, citations, and mathematical claims.

[1] PyTorch's CPU and CUDA softmax kernels (`host_softmax` and `SoftMax.cu`'s epilogue $\exp(\mathrm{input} - \max)/\mathrm{sum}$), FlashAttention's online softmax (exponentiating $\mathrm{score} \cdot \mathrm{scale} - \max \cdot \mathrm{scale}$), Triton's fused-softmax kernel ("subtract maximum for numerical stability"), and TensorFlow's softmax op (`shifted_logits = logits - max`) all evaluate the exponential only on inputs shifted by the row maximum.

REFERENCES

Dana Angluin, David Chiang, and Andy Yang. Masked hard-attention transformers recognize exactly the star-free languages. In *Proceedings of the Conference on Language Modeling (COLM)*, 2024. arXiv:2310.13897; introduces Boolean RASP (B-RASP).

Satwik Bhattamishra, Arkil Patel, and Navin Goyal. On the computational power of transformers and its implications in sequence modeling. In *Proceedings of the 24th Conference on Computational Natural Language Learning (CoNLL)*, pp. 455–475, 2020.

Moritz Brösamle and Stephan Eckstein. The expressive power of low precision softmax transformers with (summarized) chain-of-thought, 2026. arXiv:2605.18079.

Guanyu Cui, Zhewei Wei, and Kun He. Position: The Turing-completeness of real-world autoregressive transformers relies heavily on context management, 2026. arXiv:2605.19514, accepted to the ICML 2026 Position Paper Track.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Da-Wei He, and Liwei Wang. Towards revealing the mystery behind chain of thought: a theoretical perspective. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, pp. 70757–70798, 2023.

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. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*, pp. 46242–46293, 2025.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *Proceedings of the 40th International Conference on Machine Learning (ICML 2023)*, volume 202 of *Proceedings of Machine Learning Research*, pp. 11398–11442, 2023.

Xinting Huang, Andy Yang, Satwik Bhattamishra, Yash Sarrof, Andreas Krebs, Hattie Zhou, Preetum Nakkiran, and Michael Hahn. A formal framework for understanding length generalization in transformers. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025. arXiv:2410.02140.

Hongjian Jiang, Michael Hahn, Georg Zetzsche, and Anthony Widjaja Lin. Softmax transformers are Turing-complete. In *Proceedings of the 14th International Conference on Learning Representations (ICLR)*, 2026. arXiv:2511.20038.

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 *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.

Qian Li and Yuyi Wang. Constant bit-size transformers are Turing complete. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*, pp. 69254–69273, 2025a. doi: 10.52202/085713-2083.

Qian Li and Yuyi Wang. Efficient Turing machine simulation with transformers, 2025b. arXiv:2512.00003.

Qian Li, Xinyu Mao, and Shang-Hua Teng. Rethinking the role of positional encoding: Sliding-window transformers without PE remain Turing complete, 2026. arXiv:2606.01532.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*, 2024. arXiv:2402.12875.

David Lindner, János Kramár, Matthew Rahtz, Thomas McGrath, and Vladimir Mikulik. Tracr: Compiled transformers as a laboratory for interpretability. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023. arXiv:2301.05062.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.07923.

William Merrill and Ashish Sabharwal. A little depth goes a long way: The expressive power of log-depth transformers. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*, 2025. arXiv:2503.03961.

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is Turing-complete. *Journal of Machine Learning Research*, 22(75):1–35, 2021.

Emil L. Post. Finite combinatory processes — formulation 1. *Journal of Symbolic Logic*, 1(3): 103–105, 1936.

Ruizhong Qiu, Zhe Xu, Wenxuan Bao, and Hanghang Tong. Ask, and it shall be given: On the Turing completeness of prompting. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025. arXiv:2411.01992.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025. arXiv:2502.17416.

Dale Schuurmans, Hanjun Dai, and Franco Zanini. Autoregressive large language models are computationally universal, 2024. arXiv:2410.03170.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 464–468, 2018.

Lena Strobl, Dana Angluin, David Chiang, Jonathan Rawski, and Ashish Sabharwal. Transformers as transducers. *Transactions of the Association for Computational Linguistics*, 13:200–219, 2025. doi: 10.1162/tacl_a_00736. arXiv:2404.02040; extends B-RASP to transductions.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, pp. 24824–24837, 2022.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers. In *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9979–9991, 2021.

Kevin Xu and Issei Sato. A formal comparison between chain of thought and latent thought. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. arXiv:2509.25239, camera-ready version.

Andy Yang and David Chiang. Counting like transformers: Compiling temporal counting logic into softmax transformers. In *Proceedings of the Conference on Language Modeling (COLM)*, 2024. arXiv:2404.04393; introduces C-RASP.

Chenxiao Yang, Nathan Srebro, David McAllester, and Zhiyuan Li. PENCIL: Long thoughts with short memory. In *Proceedings of the 42nd International Conference on Machine Learning (ICML 2025)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 71092–71156. PMLR, 2025.

Liu Yang, Kangwook Lee, Robert D. Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*, 2024. arXiv:2311.12424.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Josh Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? a study in length generalization. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.16028; introduces RASP-L.

A Congruent Queue Lengths

Grouping L consecutive steps of a synchronous K-queue machine into a single block requires that the front of queue r at global step $Lt + l$ be the symbol written at the same intra-block offset of an earlier block. That symbol was written at step $Lt + l - \ell_r + 1$, so the alignment

$$Lt + l - \ell_r + 1 \;=\; L(t - w_r + 1) + l$$

holds for all t and $0 \le l < L$ if and only if

$$\ell_r \equiv 1 \pmod{L}, \qquad w_r = \frac{\ell_r - 1}{L} + 1.$$

In the conversion of Fact 2, level i has two content-half queues of length B^i and one buffer queue of length $2B^i$, where $B = \lceil s(n)^{1/k} \rceil$. We split the buffer into two queues of length B^i, so that all four physical queues at the level have the same length.

Lemma A.1. *Prescribe $L = L(n) \ge 1$ and choose a fixed integer $k \ge 1$ satisfying $s(n)^{1/k} = \Omega(L)$. Let $B = \lceil s(n)^{1/k} \rceil$ and*

$$b = \min\big\{\, b' \ge \max\{B,\ L+1\} \;:\; b' \equiv 1 \pmod{L} \,\big\}, \qquad \ell_i = b^{\,i} \qquad (1 \le i \le k).$$

Then the conversion can be implemented with $K = 8kk'$ queues, all four queues at level i having length ℓ_i. Every queue length satisfies $\ell_i \equiv 1 \pmod{L}$, and the resulting machine runs in $O(t(n)s(n)^{1/k})$ steps using $O(k's(n))$ queue space. Moreover,

$$w_r = \frac{\ell_r - 1}{L} + 1 \ge 2, \qquad w = O\!\left(\frac{s(n)}{L}\right).$$

Proof. Run the conversion of Fact 2 with geometric base b. Replacing B by $b \ge B$ preserves the divisibility between consecutive levels and provides at least the original capacity. Split each length-$2\ell_i$ buffer, as any queue of length $m_1 + m_2$ can be simulated step-for-step by two queues of lengths m_1 and m_2; each stack now uses four length-ℓ_i queues per level, so a k'-tape machine uses $2 \cdot 4kk' = 8kk'$ queues. The split simulation is step-for-step equivalent to the original one.

By minimality,

$$b \le \max\{B, L+1\} + L - 1 = O(s(n)^{1/k}),$$

because $L = O(s(n)^{1/k})$. Together with $b \ge B$, this gives $b = \Theta(s(n)^{1/k})$ and $\ell_k = b^k = \Theta(s(n))$. The conversion therefore takes $O(t(n)b) = O(t(n)s(n)^{1/k})$ steps and $O(k' \sum_{i=1}^{k} \ell_i) = O(k's(n))$ queue space. Finally, $b \equiv 1 \pmod{L}$ implies $\ell_i = b^i \equiv 1 \pmod{L}$, while $b \ge L+1$ gives $w_r \ge 2$; hence $w = \Theta(\ell_k/L) = O(s(n)/L)$. □

The construction introduces no additional transitions, so $\tau(n) = O(t(n)s(n)^{1/k})$, the run length used in Proposition 6 and Theorem 5.

B Transformer Components

This appendix specifies the transformer of Theorem 5 / Proposition 6 in full. Throughout, $M = (\Sigma, Q, \delta)$ is the congruent synchronous K-queue machine of Proposition 6, with a fixed alphabet Σ, $|Q| = c$, halting states normalized as absorbing (§3.1), and lengths $\ell_r = L(w_r - 1) + 1$ with

integers $w_r \geq 2$; $w = \max_r w_r$, $s_{\max} = \max_r \ell_r$, so $wL = s_{\max} - 1 + L$. The model is the looped transformer of Definition 3 with one decoder block ($m = 1$), $H = K$ heads (head r serves queue r), key dimension $d_k = K$, value dimension $d_v = |\Sigma|$, and loop count L; the same block, embedding, and output head are reused at every iteration and every pass. Fix an indexing map $\iota : \Sigma \to \{1, \ldots, |\Sigma|\}$.

B.1 Vocabulary and embedding

$\mathcal{V} = (Q \times \Sigma^K) \cup \{\text{ACCEPT}, \text{REJECT}\}$. The generated token at position i is (q_{Li}, σ_{Li}) — the state and the K-tuple appended at log index Li — or, at halting, one of the two terminal tokens (Appendix B.6).

$d = 2|\Sigma|K + c + 1$, laid out in blocks:

coordinates	block	content
0	constant	1
$1, \ldots, c$	state	one-hot e_q of the state q
$c{+}1{+}\lvert\Sigma\rvert r, \ldots, c{+}\lvert\Sigma\rvert(r{+}1)$	symbol r	one-hot of $\sigma^{(r)} \in \Sigma$
$c{+}\lvert\Sigma\rvert K{+}1{+}\lvert\Sigma\rvert r, \ldots, c{+}\lvert\Sigma\rvert K{+}\lvert\Sigma\rvert(r{+}1)$	front r	one-hot of the attention-retrieved front; 0 at embedding

$\mathrm{emb}(q, \sigma) = (1,\ e_q,\ \mathrm{onehot}(\sigma^{(0)}), \ldots, \mathrm{onehot}(\sigma^{(K-1)}),\ 0, \ldots, 0)$; terminal tokens take $\mathrm{emb}(\text{ACCEPT}) = \mathrm{emb}(q_{\text{acc}}, \#^K)$ and $\mathrm{emb}(\text{REJECT}) = \mathrm{emb}(q_{\text{rej}}, \#^K)$ (generation stops at either).

B.2 Relative positional encoding

The relative positional encoding (the form of Shaw et al. (2018); (Li & Wang, 2025b, Eq. (2))): for distance $i - j \in \{0, \ldots, w - 1\}$,

$$\mathrm{pos}(i{-}j) \;=\; \Delta \sum_{r:\, i-j=w_r-1} e_r \;\in\; \mathbb{R}^{d_k},$$

where $\Delta > 0$ is the fixed constant of §4.2 and e_r is the r-th standard basis vector. Head r retrieves offset $w_r - 1$ by the query e_r, so $\langle e_r,\, \mathrm{pos}(i{-}j)\rangle = \Delta[\, i{-}j = w_r{-}1\,]$. The encoding is unlearnable and depends on $(\ell_0, \ldots, \ell_{K-1}, L)$ only — the learnable parameter vector is one and the same for all input lengths, the analogue of (Li & Wang, 2025b, Remark 2).

B.3 The attention heads

For head $r \in \{0, \ldots, K - 1\}$, the matrices $W_q^{(r)}, W_k^{(r)} \in \mathbb{R}^{d_k \times d}$ and $W_v^{(r)} \in \mathbb{R}^{d_v \times d}$:

- **Query** $W_q^{(r)}$: single nonzero entry $W_q^{(r)}[r, 0] = 1$, copying the constant coordinate of h onto the r-th head coordinate. Then $W_q^{(r)}\, h_i^{(l)} = e_r$ for every token, every iteration.
- **Key** $W_k^{(r)} = 0$: the keys carry no content; the positional encoding $\mathrm{pos}(i - j)$ of B.2 enters the key side of the scores (§3.2). In the cache only the value entries are informative.
- **Score**: $\mathrm{score}_r(i, j) = \langle W_q^{(r)} h_i^{(l)},\ W_k^{(r)} h_j^{(l)} + \mathrm{pos}(i{-}j)\rangle = \langle e_r,\, \mathrm{pos}(i{-}j)\rangle = \Delta[\, i{-}j = w_r{-}1\,]$. Over the window $\{i{-}w{+}1, \ldots, i\}$ this is Δ at the single position $j^* = i{-}w_r{+}1$ (in range at every pass, since $i \geq w - 1$ and $w_r \leq w$) and 0 elsewhere. Self-attention has score 0 because $w_r \geq 2$. The unique maximum has gap Δ (§4.2), so hardmax returns the one-hot vector $s_{r,i,\cdot}^{(l)} = \mathbf{1}_{j^*}$.
- **Value** $W_v^{(r)}$: $W_v^{(r)}[\, \iota(\alpha),\ c + |\Sigma|r + \iota(\alpha)\,] = 1$ for $\alpha \in \Sigma$ — the projection onto symbol block r. With the write-time invariant (Lemma 4), $W_v^{(r)}\, h_j^{(l)} = \mathrm{onehot}\big(\sigma_{Lj+l}^{(r)}\big)$, hence the head output

$$a_{i,r}^{(l)} \;=\; \mathrm{onehot}\Big(\sigma_{L(i-w_r+1)+l}^{(r)}\Big) \;=\; \mathrm{onehot}\Big(\sigma_{Li+l-\ell_r+1}^{(r)}\Big),$$

the front of queue r at machine step $Li+l$ — the retrieval identity, using $\ell_r - 1 = L(w_r - 1)$.

B.4 Head combination

$\mathrm{concat}_r\, a_{i,r}^{(l)} \in \mathbb{R}^{|\Sigma|K}$; $W_O \in \mathbb{R}^{d \times |\Sigma|K}$ with $W_O[\, c + |\Sigma|K + |\Sigma|r + \iota(\alpha),\ |\Sigma|r + \iota(\alpha)\,] = 1$ for every r and $\alpha \in \Sigma$, all other entries 0; $b = 0$. The residual update $\tilde{h}_i^{(l)} = h_i^{(l)} + W_O\,\mathrm{concat}$ writes the K retrieved fronts into the front blocks; the constant, state, and symbol blocks pass through unchanged.

B.5 Feed-forward network: the transition

FF is a two-layer ReLU network of width $|Q| \cdot |\Sigma|^K + O(d)$ (a machine constant) realizing $\mathrm{FF}(\tilde{h}) + \tilde{h} = (1,\ e_{q'},\ \mathrm{onehot}(\sigma'),\ 0, \ldots, 0)$ where $(q', \sigma') = \delta(q, f)$, with q in the state block, σ in the symbol blocks, and f in the front blocks of $\tilde{h}$. For the transition lookup, define

$$u_{(\hat{q},\hat{f})}(\tilde{h}) \;=\; 2\,\mathrm{ReLU}\Big(\tilde{h}_{1+\hat{q}} + \textstyle\sum_r \tilde{h}_{c+|\Sigma|K+|\Sigma|r+\iota(\hat{f}^{(r)})} - (K+1) + \frac{1}{2}\Big) \;\in\; \{0,1\},$$

equal to 1 exactly on the table row $(\hat{q}, \hat{f})$ matching $\tilde{h}$; then

$$\begin{aligned}\mathrm{FF}(\tilde{h}) \;=\; &\sum_{(\hat{q},\hat{f})} u_{(\hat{q},\hat{f})}(\tilde{h}) \cdot \big(0,\ e_{\delta_1(\hat{q},\hat{f})},\ \mathrm{onehot}(\delta_2(\hat{q},\hat{f})),\ 0^{|\Sigma|K}\big) \\ &- \big(0,\ e_q,\ \mathrm{onehot}(\sigma),\ \mathrm{onehot}(f)\big),\end{aligned}$$

where $\mathrm{onehot}(f)$ is the concatenation of the K one-hot front blocks. The linear subtraction is implemented inside the two-layer ReLU network by $O(d)$ passthrough units: every relevant coordinate of $\tilde{h}$ is nonnegative on the invariant domain, so $\mathrm{ReLU}(\tilde{h}_a) = \tilde{h}_a$. Coordinate 0 is left unchanged. Weights and biases belong to a fixed machine-dependent set, including the threshold bias $-(K + \frac{1}{2})$; total parameter bit-size is $O((|Q||\Sigma|^K + d)d)$, constant for a fixed machine and independent of n and L. On invariant states, all intermediate sums range over a fixed finite set of half-integers, so a fixed machine-dependent numerical format suffices for exact evaluation. The absorbing convention for halting states (§3.1) is part of δ; mid-pass halting needs no extra machinery.

B.6 Output head

$W_{\mathrm{out}} \in \mathbb{R}^{|\mathcal{V}| \times d}$. For each $(q, \sigma) \in Q \times \Sigma^K$: $W_{\mathrm{out}}[(q,\sigma),\ 1+q] = 1$ and $W_{\mathrm{out}}[(q,\sigma),\ c + |\Sigma|r + \iota(\sigma^{(r)})] = 1$ for every r; terminal rows $W_{\mathrm{out}}[\mathrm{ACCEPT},\ 1+q_{\mathrm{acc}}] = W_{\mathrm{out}}[\mathrm{REJECT},\ 1+q_{\mathrm{rej}}] = K+2$. Since $h_i^{(L)}$ has exactly one nonzero per one-hot block, the ordinary row of the emitted token scores $K+1$, every other ordinary row less, and the terminal rows score $K+2$ exactly when the state block is halting (and 0 otherwise): the argmax is $(q_{L(i+1)}, \sigma_{L(i+1)})$ at non-halting passes and ACCEPT/REJECT at halting ones. Entries are $O(K)$, constant for a fixed machine.

B.7 The loop and the input interface

One block, Definition 3 with $m = 1$: $h_i^{(l+1)} = \mathrm{FF}(\tilde{h}_i^{(l)}) + \tilde{h}_i^{(l)}$ for $l = 0, \ldots, L-1$, the matrices of B.1–B.6 identical at every iteration. The per-iteration KV cache (§3.2) holds, for each position j and iteration l, the value entries $W_v^{(r)} h_j^{(l)} = \mathrm{onehot}(\sigma_{Lj+l}^{(r)})$, $r = 0, \ldots, K-1$ — the keys are zero (B.3), the positional encoding entering only the scores (B.2). The input interface is the initialization of §4.3: the value entries of positions $0, \ldots, w-2$ at all iterations contain the K-tuples $\sigma_0, \ldots, \sigma_{s_{\max}-2}$ of the log, and the current token at position $w-1$ is $(q_{s_{\max}-1}, \sigma_{s_{\max}-1})$. Generation starts at pass $w-1$.

B.8 Faithfulness

Proposition B.1. Under the initialization of §4.3, the transformer of B.1–B.7 emits, at each pass $t \geq w-1$, the token $(q_{L(t+1)}, \sigma_{L(t+1)})$, and after $\lceil \tau(n)/L \rceil$ passes the terminal token matching M's verdict on x; there is no other behavior.

Proof. Lemma 4's two structural facts hold verbatim for the components above: (a) within a pass, state moves only through the residual stream — the attention reads same-iteration value entries only, and each iteration's FF consumes the fronts its attention wrote; (b) value entries are frozen at write time. With the base case supplied by the initialization of §4.3 and the retrieval identity of B.3, the two-stage induction of §4.3 goes through unchanged; the output head of B.6 turns the final residual into the token or the verdict, and the absorbing convention makes mid-pass halting consistent. □