TEST-TIME COMPUTE IN CHAIN-OF-THOUGHT AND LOOPED TRANSFORMERS: SIMULATION AND RESOURCE TRADE-OFFS

Anonymous authors
Paper under double-blind review

ABSTRACT

Increasing test-time compute can improve the reasoning capabilities of large language models. Two prominent approaches are chain-of-thought (CoT), which generates intermediate tokens, and looped Transformers, which repeatedly update a latent workspace. We theoretically compare their execution efficiency at fixed correctness, allowing each architecture to choose its own algorithm. Our main contribution is a matching work–call trade-off for parity that accounts for workspace capacity and the concentration of access costs across rows. In a finite-precision hard-attention/ReLU model with a fixed finite alphabet for retained data, we show that, with sufficient workspace, polylogarithmic work overhead is insufficient to attain the fastest call order under balanced row costs. Concentrating costly reductions in fewer rows can attain this order at linear work. Work includes all evaluated rows, and the lower bound applies to all algorithms in the stated class. Supporting results establish mutual simulation of finite deterministic computations with explicit overhead and a linear-work separation allowing full admitted numerical states: CoT requires linearly many tokens, whereas a compact loop needs logarithmically many calls. A single-GPU study of the explicit constructions illustrates that reducing source reads does not necessarily reduce latency, with the outcome depending on batch size. Learned reasoning quality is outside the scope of this comparison.

1 INTRODUCTION

Test-time compute is the computation a language model performs at inference to produce an answer. Increasing this budget can improve reasoning while keeping learned parameters fixed (Snell et al., 2025). It allows more intermediate computation on difficult problems, but the cost of that computation depends on how it is organized.

Two prominent approaches organize it differently. *Chain-of-thought* (CoT) generates intermediate tokens that later steps can use (Nye et al., 2021; Wei et al., 2022). Each token requires a decoder evaluation and extends the sequence, which retains both token history and numerical key–value activations. A *looped Transformer* repeatedly applies a shared block to an array of internal numerical states (Dehghani et al., 2018; Geiping et al., 2026). Each application, which we call a *loop call*, updates the array's positions, or *rows*, directly in latent space.

Both approaches perform additional computation before answering, but one token step and one loop call need not do the same amount of work. This distinction motivates our two questions: *Can CoT and looped Transformers perform the same tasks, and what resources does each need to produce the same answer?*

From capability to execution cost. Prior work establishes simulation and parallel-computation capabilities (Saunshi et al., 2025; Xu & Sato, 2025). To compare execution efficiency, we must also account for what each step does. A cached CoT step evaluates one new position; a loop call updates many rows. An update that searches a large set of sources also costs more than one that reads a single address. Fewer calls can therefore require more total work. We ask how much work is necessary to achieve a target call count when workspace and the distribution of access costs across rows are constrained.

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.

A task that separates capability from cost. Our main task is parity: deciding whether an input contains an odd number of ones. The answer is a single bit, but every input bit matters. A sequential algorithm maintains a running odd/even value. A parallel algorithm computes partial answers and combines them in pairs, doubling the covered input each round. Parity thus exposes the organization of computation without requiring a long output. Parallel reduction is established; our concern is its optimal cost when every update and input access is counted. Each interface may choose its own algorithm.

A common model and accounting. We compare total operations and mutable workspace in a model with finite-precision hard attention, ReLU updates without normalization, and fixed block width, depth, and head count. Initial encoding is deterministic and has bounded depth. A certified address permits a single-source read; a search pays for every scanned slot. Preparation, storage, and unused operations count. CoT retains its numerical cache and may sample tokens or stop adaptively. Both interfaces must meet the same error bound, below one half, on every input; costs are worst-case expectations (Section 3).

For loops, we also constrain *row-cost imbalance*: how much source-reading work may concentrate in a few rows. Requiring comparable row costs makes expensive searches costly across the array; allowing them alongside cheap updates can reduce total work. This constraint is central to our main result, which characterizes the fewest calls under joint work and workspace budgets for parity with fixed-alphabet retained data.

Main findings. The results proceed from a simulation baseline to a same-work comparison and then to the optimal cost of further reducing loop calls.

1. **Mutual simulation with explicit overhead.** A loop can reproduce a finite deterministic CoT trajectory by storing its prefix; CoT can reproduce a finite deterministic loop's answer by serializing its digital operations (Theorem 3). These constructions make the costs of established simulations explicit. They may expand architecture, precision, and storage, so they establish capability without implying equal efficiency.
2. **A conditional loop advantage at linear work.** On parity, CoT needs linearly many tokens, whereas a loop achieves logarithmically many calls at linear work with sufficient workspace, certified indexed reads, and comparable row costs (Theorems 4–5; Corollary 6). The advantage comes from combining partial answers in parallel while limiting each call's work. About $n/\log n$ rows suffice for n inputs; evaluating one row per input raises work by a logarithmic factor. This comparison permits full admitted numerical states, without the frontier's retained-data restriction, and also holds for the ordered-group task studied here (Proposition 23).
3. **A matching trade-off for further reductions in calls.** Our main result gives matching lower bounds and constructions for the joint trade-off between work, workspace, and row-cost imbalance (Theorem 8). Each row's retained input-dependent data, including controller fields, must belong to a fixed finite set at initialization and after every call. The set is independent of input length and budgets; input-independent metadata may grow, and within-call computations retain the model's full numerical precision. With comparable row costs, sublogarithmic calls require more than linear work. With sufficient capacity, concentrating expensive reductions in fewer rows attains the model's fastest call order at linear work. The bounds optimize over all admitted algorithms.

Scope and limitations. The comparison concerns execution efficiency at fixed correctness on explicit algorithmic tasks. It does not predict learned accuracy, training success, or softmax behavior. The frontier for full numerical records remains open at intermediate budgets (Appendix E.5).

The results identify how workspace and the distribution of attention-access costs determine the work needed to reduce loop calls. The single-GPU study illustrates a further distinction: fewer calls or source reads need not lower latency (Section 7).

2 RELATED WORK

The closest results compare expressivity, sequential steps, and memory. Table 1 locates our work among them; our focus is the optimal work–call trade-off under joint workspace and row-cost constraints.

Table 1: Selected theoretical comparisons. CT denotes continuous thought. Attention, precision, and uniformity assumptions differ across papers; our matching frontier assumes the stated state class and execution rules.

Paper	Models	Central result	Resource focus
Merrill & Sabharwal (2024)	CoT	Expressivity as CoT length grows	Token budgets; complexity classes
Saunshi et al. (2025)	CoT, Loop	CoT simulation; group composition	Loops and effective depth
Amiri et al. (2025)	CoT	CoT lower bounds, including parity	Required token count
Merrill & Sabharwal (2026b)	Padded TF, Loop	Uniform parallel-class characterizations	Padding and loop depth
Xu & Sato (2025)	CoT, CT, Loop	DAG evaluation; counting and sampling	Iterations; size–depth tradeoffs
Svete & Sabharwal (2026)	CoT, Loop, Masked	Padded-loop and masked-diffusion simulations	Steps and padding; log-width families
Li et al. (2026)	CoT and variants	Word RAM simulation	Algorithmic simulation overhead
Zhang (2026)	CoT, Loop	Conditional compressed-state separation	Persistent-memory budgets
Ours	**CoT, Loop**	**Matching parity work–call bounds**	**Work, calls, capacity, imbalance**

Expressivity and sequential computation. Existing comparisons provide the capability and step-count baselines. Amiri et al. (2025) prove deterministic CoT lower bounds for parity; their simultaneous prefill restriction underlies our bound for sampled decoding with numerical KV records and adaptive stopping. Svete & Sabharwal (2026) connect CoT and padded loops through masked diffusion with finite precision and logarithmic width, while Xu & Sato (2025) compare CoT and latent thought through computation graph size and depth. Under different architectural assumptions, our frontier additionally constrains executed work, workspace, and row-cost imbalance. It charges every evaluated row and physical scan slot, including unused operations, whose cost graph depth alone does not give.

Memory and execution budgets. Resource constraints lead to different comparisons. Zhang (2026) obtains a conditional compressed-state separation under persistent-memory budgets, and Yang et al. (2024) analyze model-size requirements for dynamic-programming tasks with sparse and linear attention. We fix architecture caps and charge execution, including frozen-input access, for tasks both interfaces can solve.

Constructions and lower-bound tools. Our analysis draws on programmable loops (Giannou et al., 2023), constructive CoT algorithms (Feng et al., 2023; Li et al., 2024), and log-depth computation (Merrill & Sabharwal, 2026a). The lower bounds combine attention limitations and circuit representations (Hahn, 2020; Hao et al., 2022), contextual queries (Barceló et al., 2026), and formula sensitivity and criticality (Rossman, 2018; Harsha et al., 2023). Classical CREW parity bounds support the parallel-time result (Dietzfelbinger et al., 1994; Kutyłowski, 2005). Softmax parity constructions (Chiang & Cholak, 2022) use a different attention model from our lower bounds.

3 A COMMON COMPUTATIONAL MODEL

Both interfaces receive $x \in \{0,1\}^n$ and return $f_n(x) \in \{0,1\}$, choosing their own algorithms under the same correctness requirement. We compare work and steps separately, focusing on parity, $f_n(x) = x_1 \oplus \cdots \oplus x_n$; Section 5 also treats ordered finite-group composition with constant-length binary codes.

3.1 TWO WAYS TO MAINTAIN AN EVOLVING COMPUTATION

CoT appends causal positions; a loop revises an existing array of positions, called rows. The numerical and control fields at a position form its *record*.

Definition 1 (The two interfaces)**.** A CoT decoder generates $z_{t+1} = \mathrm{decode}_\theta(x, z_{1:t})$ using the same causal Transformer at each step; its final token determines the answer. Its physical state includes the tokens and all retained layerwise KV records. Positions and past causal activations are prefix-consistent: appending a token does not alter them. The decoder has no separate recurrent latent buffer. Randomized decoding follows one trajectory with one fixed-depth evaluation per newly processed position.

A loop iterates a shared block

$$H^0 = E(x), \qquad H^{r+1} = Q_p F_\theta(H^r, E_x(x)), \qquad H^r \in \mathbb{F}_p^{m \times d_L},$$

where the m mutable rows remain fixed during a run. The optional frozen input representation $E_x(x)$ is charged. A fixed affine binary-score head reads a designated row. The finite format $\mathbb{F}_p$, dynamic range, rounding, and overflow rules are specified; Q_p denotes the declared digital arithmetic. Growing-sequence latent feedback is outside these interfaces.

Thus CoT retains numerical state as well as tokens, but cannot revise old KV records. A loop overwrites its rows; using fewer mutable rows does not remove frozen-input storage from the budget.

3.2 ARCHITECTURE AND EXECUTION ASSUMPTIONS

Architecture and input interface. Width, block depth, and head count are fixed constants. Each complete record occupies $O(\log n)$ bits, with a coefficient fixed as budgets vary. Attention selects one source with a fixed total tie order; positionwise maps use the admitted finite-precision ReLU operations without normalization. A deterministic encoder with fixed depth/head caps produces $O(n)$ frozen records and initial mutable records, each combining constantly many encoder activations at fixed positions. Positionwise embeddings are allowed, as are input-independent constants and padding added after encoding. No extra encoder positions or uncharged global pooling are supplied. Encoding, positional-field construction, and preprocessing are charged.

Work and local execution. Each head declares its access mode: a *gather* or a *scan*. A gather reads one source, certified by the integer-position score $2aj - j^2$ on an interval (the clamped address a) or a fixed singleton selector. A scan visits a source list, or bank, and counts every physical slot, including masked and repeated slots. Declared kernels execute even if an equivalent computation would be cheaper. Address construction, certification, and source-list access are charged. Each word primitive accesses constantly many separately addressed records; raw bits initially occupy separate records.

Where routing decisions come from. Routing obeys the same local accounting. At stage s, a destination's preceding record, designated controller records D with $|D| \le d_0$, fixed input-independent metadata, and independent coins determine its kernels and complete ordered source lists. Descriptor enumeration cannot inspect other input-dependent records to discover source identities. Data selecting a later list must first enter its destination or controllers through an earlier admitted, charged operation; no hidden state is allowed. Scores and masks use declared query/source records and metadata. Stopping and readout use fixed local rules on constantly many designated records, with constantly many checks per stage. CoT stopping may inspect computed token states but has no external oracle into the frozen cache.

How much row costs may differ. All rows and declared heads execute, including unused outputs; their costs can differ with source-list length. For $k_{sih} = 1$ at a gather and the bank size at a scan, put

$$a_{si} = 1 + \sum_h k_{sih}, \qquad \kappa_s^{\text{eff}} = \frac{m \max_i a_{si}}{\sum_i a_{si}} \le \kappa.$$

The cap κ bounds the largest row access charge relative to the stage average, not processor count. Section 5 fixes $\kappa = \kappa_0$; Section 6 allows an integer cap $1 \le \kappa \le M_{\text{row}}$ to grow; larger caps are redundant since $\kappa_s^{\text{eff}} \le m \le M_{\text{row}}$. Update work is at least $a_0 \sum_{s,i} a_{si}$ for a fixed $a_0 > 0$ independent of κ; costly maps or descriptors add their work.

For example, four single-head rows scanning banks of lengths $16, 1, 1, 1$ have charges $17, 2, 2, 2$: total 23 and imbalance $68/23$. Padding all banks to 16 slots makes imbalance one but raises the total to 68. Masked slots add work without information. Partial padding gives intermediate caps while preserving all-row execution (Lemma 16). Dense all-to-all scans are balanced but pay quadratic scan costs; the optima below do not determine the optimum under that policy.

3.3 What is fixed by a budget?

Definition 2 (Resources and task optima)**.** An implementation has total finite-word work C, peak physical memory M in bits, primitive critical-path span D_{span}, and parameter/program size P in bits. Budgets constrain these separately. Tokens, KV caches, input encodings, latent states, resident weights, expanded positional tables, and temporary buffers all count toward memory. Program preparation is also charged. Layer count and call count do not by themselves determine span or hardware time; growing-precision bit work would require an additional implementation analysis.

For randomized task algorithms, coins are independent of the input and used after deterministic prefill. Require almost-sure termination and $\Pr[A(x) \ne f_n(x)] \le \delta$ for every x, with fixed $\delta < 1/2$. Write $\bar{C} = \sup_x \mathbb{E}C(x)$ and $\bar{R} = \sup_x \mathbb{E}R(x)$, and similarly $\bar{T}$ for tokens. Each optimum is the infimum over all admitted implementations satisfying the specified constraints, with architecture caps fixed beforehand. Our main slice minimizes calls under work budget B, row capacity $m \le M_{\text{row}}$, and $O(n \log n)$ stored and program bits.

Costs average over independent coins before taking the worst input; correctness must hold for every input. Capacity is an upper bound: the loop chooses $m \le M_{\text{row}}$ and executes all m rows on every call. The same-work results allow full numerical records; the work–call frontier adds the finite-payload condition introduced below.

4 Mutual simulation

We first make the costs of established simulations explicit: loops maintain token prefixes, while CoT evaluates digital circuits (Saunshi et al., 2025; Li et al., 2024; Merrill & Sabharwal, 2024).

Theorem 3 (Exact mutual simulation with charged overhead)**.** *Consider deterministic models with hard attention, ReLU feedforward layers, no normalization, and specified finite-precision arithmetic. Let the binary input be nonempty and the computation horizons be positive and finite.*

(i) ***CoT to loop.*** *A prefix-consistent digital decoder with L layers, width d, precision p, and vocabulary size V, generating T tokens on an n-token input, has an exact trajectory simulation by a shared loop with $N = n + T$ rows and T iterations. Its width, block depth, and precision satisfy*

$$W = O(d + V), \qquad K = O(L + \log V + 1), \qquad p' = p + O(\log(N + V + 1)).$$

Apart from charged initialization, dense word work is $O(TK(NW^2 + N^2W))$.

(ii) ***Loop to CoT.*** *If the loop's admitted initialization, transition, and Boolean readout have NAND-circuit sizes G_0, G, G_{out}, its answer after R iterations has an exact output simulation using at most*

$$T_{\text{sim}} = G_0 + RG + G_{\text{out}} + 2$$

binary tokens. The causal decoder has constant depth and width, two hard-attention heads, and $O(\log N_c)$ precision, where $N_c = n + 2 + T_{\text{sim}}$. Inference work is $O(N_c^2)$ with dense attention

and $O(N_c)$ with certified positional gathers. Its parameter storage and data workspace are each $O(N_c \log N_c)$ bits. The positional program is independent of input values; its storage and compilation are charged.

Both constructions preserve the final Boolean answer; the first also preserves every generated token. They may increase resources and leave the subclasses used for the efficiency results.

Proofs and full accounting appear in Appendix A, Lemmas 9 and 10. The reverse simulation's token cost includes evaluating the transition circuit on every call (Proposition 11). For the task comparisons that follow, each interface instead chooses its own algorithm for the final answer.

5 THE SAME TASK AT THE SAME WORK ORDER

Mutual simulation establishes capability with overhead. We now compare the best algorithms for the same task, holding the work order fixed. Throughout this section, the imbalance cap κ_0 is constant, full numerical records are allowed, and preprocessing and readout are charged. The additional retained-state condition of Section 6 is not needed here.

5.1 SEQUENTIAL COSTS AT LINEAR WORK

CoT computes parity by maintaining a running value as it incorporates input bits. The next theorem shows that numerical caching, sampling, and adaptive stopping cannot improve the order of its token count.

Theorem 4 (CoT token requirement)**.** *Under the fixed architecture and deterministic prefill assumptions, every sampled single-trajectory CoT computing parity with per-input error at most $\delta < 1/2$ requires $\bar{T} = \Omega(n)$ tokens. This permits the full contextual numerical KV cache and local adaptive stopping. An exact constant-width decoder attains $T = \Theta(n)$ and $C = \Theta(n)$.*

Proofs and implementations are in Appendices B and C.

A loop can instead combine partial answers in parallel. Its cost depends on the workspace: every row is updated and charged, including rows that do not supply the final answer.

Theorem 5 (Work and calls for a fixed loop workspace)**.** *For $1 \le m \le n$, the optimal parity work is*

$$C^*_{\mathsf{CoT}}(n) = \Theta(n), \qquad C^*_{\mathsf{Loop}}(n, m) = \Theta(n + m \log n).$$

Every admitted loop with finite expected resources obeys

$$\bar{R} \log\left(1 + \frac{\bar{C}}{m\bar{R}}\right) = \Omega(\log n). \tag{1}$$

At optimal-order work, the fewest loop calls are $\Theta(n/m + \log n)$. Exact constant-width loops attain the upper bounds with $O(n \log n)$ stored and program bits.

Parallelism has a work cost. The theorem separates the cost of incorporating the input, n, from the additional cost of evaluating the array, $m \log n$. More rows allow more partial answers to be updated together, but also raise work per call. Equation (1) quantifies this trade-off: fewer calls require more work per row within each call. Appendices B and C give the lower bounds and matching constructions.

Figure 1 illustrates the construction. The loop forms four partial parities and combines them in two rounds.

5.2 CHOOSING HOW MUCH LATENT WORKSPACE TO USE

The $m \log n$ work term suggests that the loop should not always fill its available workspace. Choosing how many rows to evaluate yields the following comparison at linear work.

CoT: one new running parity per token

$$z_1 \longrightarrow z_2 \longrightarrow \cdots \longrightarrow \boxed{z_8}, \qquad z_t = \bigoplus_{j=1}^{t} x_j.$$

Loop: four rows update on every call

	1	2	3	4
row 1	x_1	a_1	$a_1 \oplus a_2$	$\boxed{p}$
row 2	x_2	a_2	$a_2 \oplus a_3$	p
row 3	x_3	a_3	$a_3 \oplus a_4$	p
row 4	x_4	a_4	$a_4 \oplus a_1$	p

Figure 1: The same answer with different dependencies ($n = 8$, $m = 4$). Here $a_i = x_i \oplus x_{i+4}$ and $p = a_1 \oplus \cdots \oplus a_4$. All loop rows execute even though only the boxed row is read out. The display omits charged control, preparation, and unused phase computations.

Corollary 6 (Linear work with a chosen workspace)**.** *For sufficiently large fixed K and row capacity $1 \le M_{\mathrm{row}} \le n$, the fewest loop calls at work at most Kn are*

$$\Theta(n/M_{\mathrm{row}} + \log n).$$

Choosing $m = \Theta(\min\{M_{\mathrm{row}}, n/\log n\})$ attains the bound. CoT requires $\Theta(n)$ tokens at the same work order.

Below capacity $n/\log n$, input incorporation dominates the call count. Once this capacity is available, about $n/\log n$ rows suffice for logarithmic calls at linear work. Evaluating all n rows instead costs $\Theta(n \log n)$ work, with no improvement in the optimal call order. The frozen input remains stored and charged (proof in Appendix B).

Beyond parity. The same comparison extends to a reduction in which input order matters: given elements of S_5, decide whether their ordered product is the identity. CoT again needs $\Theta(n)$ expected tokens; balanced loops attain $\Theta(n/M_{\mathrm{row}} + \log n)$ calls at linear work (Proposition 23, Appendix H). Here the loop combines contiguous chunks in order. This application to an established task (Saunshi et al., 2025; Xu & Sato, 2025) extends the same-work comparison to noncommuting inputs; a group budget frontier is not established.

6 Work, calls, and row-cost imbalance

The linear-work comparison leaves a further question: what does it cost to reduce loop calls below logarithmic order? We now allow more total work, greater concentration of expensive operations in fewer rows, or both. CoT still requires $\Theta(n)$ parity tokens at every $B \ge Kn$ under the fixed architecture. For loops, even unrestricted numerical records face a call floor of $g_n = \log n/\log\log n$ for sufficiently large n (Theorem 14). Our goal is to determine how the work and imbalance budgets govern progress toward this floor.

6.1 What may be retained between calls

To obtain a matching frontier, we restrict the information retained between calls. Each row has a fixed number of possible data values, independent of input length and budgets. Input-independent addresses and clocks may still grow with input size, and arithmetic within a call retains the model's full precision.

Definition 7 (Finite retained payload)**.** Fix a constant β. At every reached initialization or completed-call boundary r, a program in $\mathcal{F}_\beta$ has complete row record $H_i^r = (\mu_i^r, v_i^r)$. The metadata μ_i^r depend only on the program, row, boundary, and independent tape; the data v_i^r belong to a prescribed alphabet of size at most 2^β. The fields and alphabets are specified before the input and work budget. All retained scratch and controller fields are included. Intermediate computations within a call may use the full admitted numerical precision.

A parity accumulator illustrates the distinction: its parity is one data bit, while its position and phase are metadata. Routing from metadata and applying finite-table updates with cleared scratch preserve this class, as does the ordered-group construction (Lemma 24). Attention may also retain a winner's payload without its address; retaining arbitrary winner indices or numerical scan histories can leave the class.

6.2 The optimal work–call trade-off

We can now state the frontier. To isolate the value of additional work, keep the retained-data alphabet fixed as the work budget varies. Choose $\beta \geq \beta_*$, where the fixed level β_* and architecture caps suffice for the construction. Let $R^*_{\mathrm{fin},\beta,\kappa}(n, B; M_{\mathrm{row}})$ be the fewest worst-case expected calls for parity among programs in $\mathcal{F}_\beta$, with $m \leq M_{\mathrm{row}}$, expected work at most B, imbalance at most κ, and $O(n \log n)$ stored and program bits. The class parameters, including the integer cap $\kappa = \kappa(n)$, are fixed independently of B.

Theorem 8 (Work and imbalance frontier)**.** *For fixed $\beta \geq \beta_*$, sufficiently large fixed K_0, $B \geq K_0 n$, and $1 \leq \kappa \leq M_{\mathrm{row}} \leq n$,*

$$R^*_{\mathrm{fin},\beta,\kappa}(n, B; M_{\mathrm{row}}) = \Theta\left(\frac{n}{M_{\mathrm{row}}} + \max\left\{ g_n, \frac{\log n}{\log(2 + \log(2 + \kappa B/n))} \right\} \right).$$

The lower bound permits per-input error $\delta < 1/2$; exact deterministic loops attain the upper bound. Constants may depend on the fixed state, architecture, precision, and error parameters, but not on $n, B, M_{\mathrm{row}}, \kappa$.

The bound separates three constraints. Capacity limits how quickly the input can be incorporated, giving n/M_{row}. The fixed architecture imposes the floor g_n. The remaining term determines how work and imbalance affect the call count. Its nested logarithm means that calls fall slowly as resources increase. Work overhead B/n and imbalance κ enter through their product, but govern different choices: how many operations are available and where they may be concentrated. At constant imbalance and linear work, the formula recovers the preceding comparison. Its constants remain uniform as κ grows.

This matching characterization relies on the retained-data restriction in Definition 7; Appendix E gives the proof. The corresponding optimum with unrestricted numerical state remains open at intermediate budgets (Appendix E.5).

6.3 What work concentration buys

Table 2 makes the frontier concrete. Combining more partial answers per call uses larger banks. When row costs must be comparable, this expense spreads across the array. A larger imbalance cap permits a few large banks alongside many small ones.

Work B	Imbalance cap κ	CoT tokens	Loop calls
Kn	1	$\Theta(n)$	$\Theta(\log n)$
$n(\log n)^a$	1	$\Theta(n)$	$\Theta(\log n / \log\log\log n)$
Kn	$\lceil (\log n)^a \rceil$	$\Theta(n)$	$\Theta(\log n / \log\log\log n)$
$n \exp((\log n)^\alpha)$	1	$\Theta(n)$	$\Theta(g_n)$
Kn	$\lceil \exp((\log n)^\alpha) \rceil$	$\Theta(n)$	$\Theta(g_n)$

Table 2: Same-task sequential optima for $M_{\mathrm{row}} = n$, fixed $a > 0$, $0 < \alpha < 1$, sufficiently large K, n, and fixed $\mathcal{F}_\beta$ with $\beta \geq \beta_*$. Here $g_n = \log n / \log\log n$. Loops choose their evaluated workspace; tokens and calls need not have equal duration.

The second and third rows show that increasing either work or permitted imbalance by a fixed power of $\log n$ gives the same call order. Even increasing both at these rates leaves calls above the floor g_n. Reaching that floor requires a larger increase: the last two rows show that a still subpolynomial increase in either resource suffices. CoT's token optimum remains linear throughout.

The construction realizes this trade-off through partial padding. It pads each bank to at least $\lceil q/\kappa \rceil$ slots, where q bounds that stage's bank lengths. Combining at most b inputs per reduction costs

$O(m(1 + 2^b/\kappa))$ per call: the base charge updates every row and the remaining term pays for bank visits. Larger κ reduces padding without changing the logical reduction; masked slots and unused heads remain charged (Lemma 16). Without any imbalance restriction, linear work reaches $\Theta(n/M_{\text{row}} + g_n)$ calls even with full numerical records (Proposition 20, Appendix F).

7 Validation Study

Research questions. We ask: (1) How does workspace size trade calls against source reads? (2) Can relaxing row-cost balance reduce reads without changing the computation? In both cases, do fewer reads or calls mean lower GPU latency?

Experimental setup. We evaluate the parity constructions with custom CUDA operators on one RTX 5090 D v2, without training. We compare cached CoT, a compact loop with about $n/\log_2 n$ rows, and a full loop with n rows, for $n = 64, 128, \ldots, 4096$. At $n = m = 4096$, we also vary reduction arity $b \in \{2, 4, 6\}$ and row-cost imbalance cap $\kappa \in \{1, 4, 32\}$. Both use batches 1 and 32, giving 60 configurations. Source visits count physical slots read, including masked and dummy slots. Timings cover complete warmed, device-resident requests under CUDA Graph replay. Appendix J gives full methods and results.

Results and interpretation. All configurations pass the correctness checks. For workspace, the compact loop uses 24 calls and 14.8% of the full loop's source visits at $n = 4096$; the full loop uses 13 calls, whereas CoT generates 4096 tokens (Table 3). This illustrates the constructions behind Theorem 5 and Corollary 6: every workspace row is updated at each call, so fewer rows can save reads despite requiring more calls.

Program	Tokens/calls	Visits/example	Batch 1 (ms)	Batch 32 (ms)
Cached CoT	4,096	8,192	4.6407	5.4341
Compact loop	24	16,384	0.0446	0.0482
Full loop	13	110,592	0.0270	0.1105

Table 3: Programmed parity at $n = 4096$. Visits include input encoding; timings are whole-batch medians of 15 block means (three input suites, five blocks each), not per-example latency or independent device replications.

For padding, increasing κ from 1 to 32 at $b = 6$ removes 87.0% of source visits while preserving the logical computation and all 37 calls. As in Lemma 16, allowing unequal row costs removes padding reads without adding steps. Latency nevertheless rises by 9.9% at batch 1 and falls by 16.2% at batch 32; the two binary loops also reverse their latency ranking across batches. Resource savings therefore need not yield GPU speedups. Each CoT token or binary loop call uses one kernel, and fused alternatives are untested. Visits do not measure complete work or FLOPs, and these finite checks do not test asymptotic bounds or optimality.

8 Discussion

On parity and our ordered-group task, loops have a conditional execution advantage: logarithmically many calls at linear work, compared with linearly many CoT tokens. This requires sufficient workspace, certified indexed reads, and comparable row costs under our hard-attention and input-encoding assumptions. Parallel updates provide the benefit because their work remains controlled.

The matching frontier characterizes how work and permitted cost concentration reduce calls at a given workspace budget under the finite-payload condition. Extending it to the model's full numerical records at intermediate budgets remains open (Appendix E.5). Primitive execution time also depends on the available processors (Corollary 22), while GPU latency depends on the execution policy (Section 7).

Training and generalization are natural directions for future work. Can training discover resource-efficient computations in each paradigm? Do they generalize to longer inputs or larger inference budgets? Answering these questions would connect execution efficiency to learning and reuse.

AI USE DISCLOSURE

Generative AI tools assisted with drafting and polishing portions of the manuscript and preparing verification code. The authors remain responsible for all claims and proofs. Finite computational checks do not substitute for mathematical proofs.

REPRODUCIBILITY STATEMENT

The appendix provides constructions and proofs of our results. The source package includes finite checks for the compilers, contextual input queries, parity constructions, local formula templates, partial padding, unequal scan banks, and ordered group products. The GPU study is documented in Appendix J. Extracted measurement records and the script generating the empirical figures and main table accompany the source package. Anonymous repository: `https://anonymous.4open.science/r/CoT-LT-5249/README.md`.

REFERENCES

Alireza Amiri, Xinting Huang, Mark Rofin, and Michael Hahn. Lower bounds for chain-of-thought reasoning in hard-attention transformers. In *arXiv preprint arXiv:2502.02393*, 2025.

Pablo Barceló, Alexander Kozachinskiy, and Tomasz Steifer. Ehrenfeucht-haussler rank and chain of thought. *Journal of Machine Learning Research*, 27(185):1–41, 2026.

Paul Beame and Johan Hastad. Optimal bounds for decision problems on the crcw pram. *Journal of the ACM (JACM)*, 36(3):643–670, 1989.

Richard P Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM (JACM)*, 21(2):201–206, 1974.

David Chiang and Peter Cholak. Overcoming a theoretical limitation of self-attention. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7654–7664, 2022.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *arXiv preprint arXiv:1807.03819*, 2018.

Martin Dietzfelbinger, Mirosław Kutyłowski, and Rüdiger Reischuk. Exact lower time bounds for computing boolean functions on crew prams. *Journal of Computer and System Sciences*, 48(2): 231–254, 1994.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: a theoretical perspective. In *Advances in neural information processing systems*, volume 36, pp. 70757–70798, 2023.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian 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*, volume 38, pp. 41340–41391, 2026.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. *International Conference on Machine Learning*, pp. 11398–11442, 2023.

Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 2020.

Yiding Hao, Dana Angluin, and Robert Frank. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Transactions of the Association for Computational Linguistics*, 10:800–810, 2022.

Prahladh Harsha, Tulasimohan Molli, and Ashutosh Shankar. Criticality of ac-formulae. In *38th Computational Complexity Conference (CCC 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

Mirosław Kutyłowski. Complexity of boolean functions on prams-lower bound techniques. In *Data structures and efficient algorithms: Final Report on the DFG Special Joint Initiative*, pp. 309–329. Springer, 2005.

Yanhong Li, Anej Svete, Ashish Sabharwal, and William Merrill. Efficiently representing algorithms with chain-of-thought transformers. *arXiv preprint arXiv:2606.19697*, 2026.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. In *International Conference on Learning Representations*, volume 2024, pp. 11911–11943, 2024.

Will 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*, volume 38, pp. 95315–95339, 2026a.

Will Merrill and Ashish Sabharwal. Exact expressive power of transformers with padding. In *Advances in Neural Information Processing Systems*, volume 38, pp. 112497–112524, 2026b.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. In *International Conference on Learning Representations*, volume 2024, pp. 7690–7706, 2024.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models, 2021.

Benjamin Rossman. The average sensitivity of bounded-depth formulas. *computational complexity*, 27(2):209–223, 2018.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, volume 2025, pp. 14855–14881, 2025.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 10131–10165, 2025.

Anej Svete and Ashish Sabharwal. On the reasoning abilities of masked diffusion language models. In *International Conference on Learning Representations*, volume 2026, pp. 139646–139681, 2026.

Uzi Vishkin and Avi Wigderson. Trade-offs between depth and width in parallel computation. *SIAM Journal on Computing*, 14(2):303–314, 1985.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in neural information processing systems*, volume 35, pp. 24824–24837, 2022.

Kevin Xu and Issei Sato. A formal comparison between chain of thought and latent thought. In *arXiv preprint arXiv:2509.25239*, 2025.

Kai Yang, Jan Ackermann, Zhenyu He, Guhao Feng, Bohang Zhang, Yunzhen Feng, Qiwei Ye, Di He, and Liwei Wang. Do efficient transformers really save computation? In *arXiv preprint arXiv:2402.13934*, 2024.

Haozhou Zhang. Chain-of-thought and compressed looped transformers: A memory-budget separation. *arXiv preprint arXiv:2605.30757*, 2026.

APPENDIX CONTENTS

Reading guide and result dependencies **13**

The frontier and its implications **14**

A Architectural simulation proofs **15**

A.1 CoT to loop . . . 15

A.2 Loop to CoT . . . 16

B Lower bounds for tokens, loop calls, and work **17**

B.1 A common query argument for contextual inputs . . . 17

B.2 Sampled CoT with its complete contextual cache . . . 18

B.3 The joint work–call lower bound . . . 19

B.4 A call floor for unrestricted numerical records . . . 20

C Exact parity implementations **21**

D Matching construction for the budget frontier **22**

E Direct proof of the finite-payload frontier **25**

E.1 Proof overview . . . 26

E.2 Local updates as formulas . . . 26

E.3 Parity and formula size . . . 27

E.4 A direct proof for finite retained payload . . . 27

E.5 An open case for unrestricted numerical state . . . 30

F The boundary of the row-cost balance assumption **30**

F.1 When row costs may differ arbitrarily . . . 30

G Parallel execution time **32**

H An ordered-group task at the same work order **32**

H.1 Finite-payload preservation . . . 33

H.2 Exact primitives and charged input decoding . . . 34

H.3 A compact loop preserving the product order . . . 34

H.4 A causal CoT implementation and the lower-bound transfer . . . 35

I Assumptions across results **36**

J Finite execution study: methods and complete comparisons **36**

J.1 Programs, inputs, and correctness checks . . . 36

J.2 What is counted and timed . . . 37

J.3 Workspace and padding results . . . 39

J.4 Backend sensitivity and reproducibility . . . 39

READING GUIDE AND RESULT DEPENDENCIES

The diagrams show how the results depend on one another and guide the reader through their proofs. Definitions 1–2, the common execution assumptions, and the prefill restriction in Appendix B.1 are understood where applicable.

How to read the arrows. A solid arrow points from a bound, condition, or unnumbered argument to the result that uses it. A dashed arrow marks reuse of a proof or explicit construction; a dotted arrow marks an optional cost instantiation. Shaded boxes are main-text results. Result numbers are links to their statements.

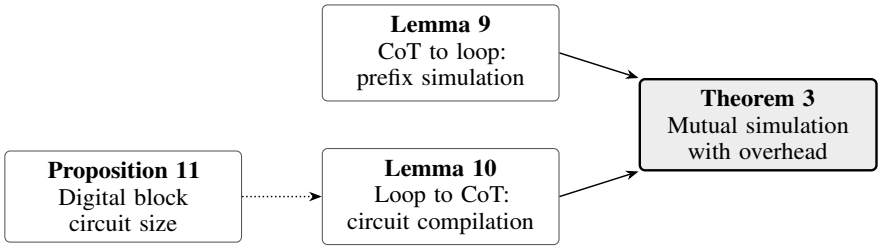


Figure 2: Mutual simulation (Appendix A). The two lemmas establish the two directions of Theorem 3. Proposition 11 makes the reverse simulation's circuit cost explicit under its primitive assumptions; the conditional compiler does not require this particular bound. The efficiency results below do not use the simulation theorem.

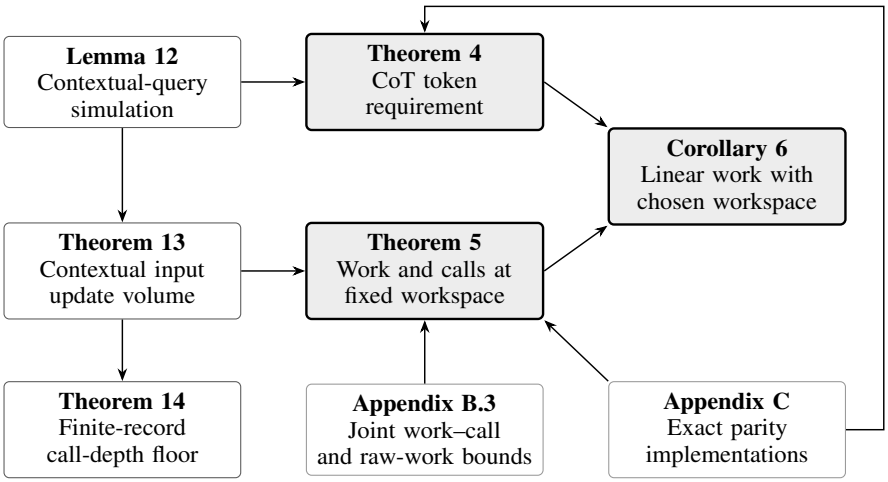


Figure 3: The same-work comparison (Appendices B–C). Lemma 12 supports both the CoT token bound and the loop input-volume bound. Theorem 14 adds an independent circuit-depth argument to the volume bound; it is not deduced from Theorem 5. The constructions supply matching upper bounds for Theorems 4 and 5.

Reading order. For the same-work comparison, read the input argument in Appendix B.1, the lower bounds in Appendices B.2–B.3, and the implementations in Appendix C. The sharper frontier uses the separate construction and formula arguments shown next.

THE FRONTIER AND ITS IMPLICATIONS

Theorem 8 combines a lower-bound argument with a matching construction. The final three comparisons clarify its scope: unequal row costs, primitive execution time, and an ordered-group task. They do not follow merely by substituting parameters into that theorem.

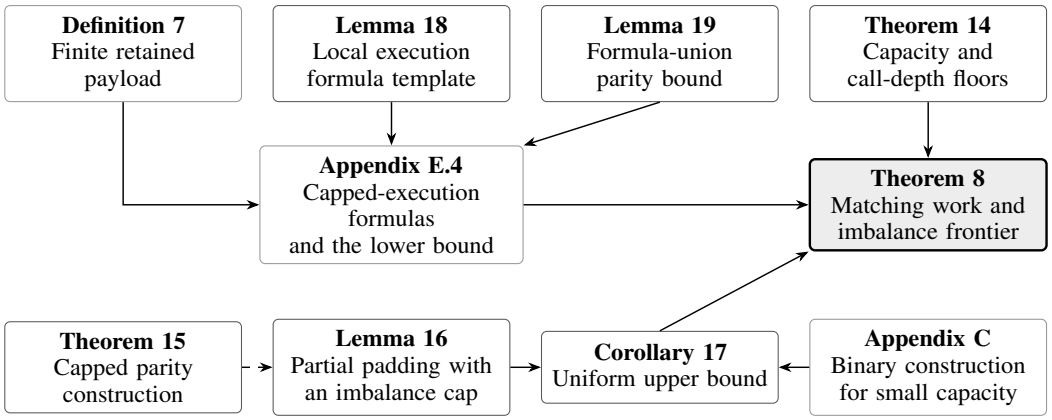


Figure 4: The matching frontier (Appendices D–E). The finite-payload condition controls the formula argument; the finite-record theorem supplies independent floors. The construction branch modifies the capped construction to meet the imbalance budget. The binary construction handles small workspace capacities. The fixed-data invariant of this upper construction is checked directly in Appendix E.4, without Lemma 24.

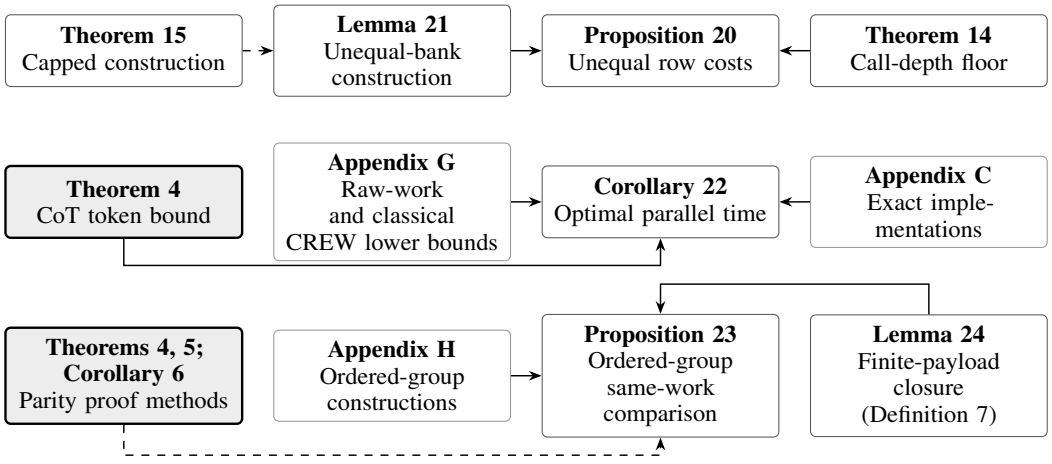


Figure 5: Three comparisons with separate proofs (Appendices F–H). Dashed arrows change the scan banks or transfer the parity arguments through an input restriction. Lemma 24 certifies only the group construction's finite-payload membership. Neither the parallel-time result nor the group comparison requires Theorem 8; the parallel-time proof also does not use Proposition 20.

The unnumbered numerical-state question in Appendix E.5 uses Theorem 14 for its lower bound and Corollary 17 for its upper bound. The observation with all n rows fixed uses Theorem 5.

A ARCHITECTURAL SIMULATION PROOFS

The two lemmas below establish the respective directions of Theorem 3, with their full constructions and resource accounting.

A.1 COT TO LOOP

Digital decoder specification. A digital decoder consists of L normalization-free residual Transformer layers of width d, a constant number of causal hard-attention heads with prescribed tie rules, and positionwise ReLU networks of width $O(d)$. Token and positional embeddings are defined through the declared horizon. Each operation specifies its quantization, rounding, and overflow behavior. Weights, activations, logits, and intermediate accumulators are signed integers with at most p bits. A linear vocabulary head returns the smallest index maximizing its logit. The implementation is *prefix-consistent*: evaluating the first i rows inside a padded sequence gives exactly the length-i computation, including its arithmetic reduction order.

Lemma 9 (Prefix simulation)**.** *For a digital decoder and $n, T \geq 1$, set $N = n + T$, $W = O(d + V)$, $K = O(L + \log V + 1)$, and $p' = p + O(\log(N + V + 1))$. A shared hard-attention/ReLU macro-block of depth K and width W on N rows reproduces its first T continuation tokens in T iterations. Its masks are fixed, and its parameters depend on the source and horizon, not input values. Apart from charged input encoding, dense work is $O(TK(NW^2 + N^2W))$. Peak data memory with materialized scores is $O((NW + N^2)p')$ bits, and parameter/controller storage is*

$$O\big(P_{\mathrm{src}} + KW^2p' + N\log(N+1)\big)$$

bits, where P_{src} includes source embeddings and positions. Resident parameters are additional to data memory. Network depth is TK; primitive-operation span depends on the evaluation algorithm.

Proof. Each row retains its embedding, original positional representation, one-hot token indicator, address and squared address, and a copy of a cursor a. Allocate $O(d + V)$ scratch channels. Initially the first n rows hold the input, the others hold fixed padding, and $a = n$. The one-hot indicators preserve token identity even if embeddings are noninjective. Protected residual channels are never normalized.

At each iteration, rebuild the source input in scratch channels and copy its L decoder layers there. Keep every source quantizer and reduction order unchanged. Signed copies and resets use two ReLU units per coordinate. Prefix consistency makes the cursor row's final scratch vector exactly the source hidden vector for the current prefix; future padded rows do not affect it. Their computation is still charged.

An all-to-all hard-attention head broadcasts that vector using

$$q(a) = (2a, -1), \qquad k(i) = (i, i^2), \qquad q(a)^\top k(i) = a^2 - (i-a)^2.$$

The maximum is uniquely at $i = a$, with integer gap at least one. These are unscaled scores; any prescribed scaling must preserve their exact ordering. The mask itself never changes. Each row applies the copied vocabulary projection and quantization. For integer logits $\ell_1, \ldots, \ell_V$, a balanced tournament using $\max(u, v) = \mathrm{ReLU}(u - v) + v$ computes their maximum m in $O(\log V)$ layers and $O(V)$ width. Define

$$e_j = \mathrm{ReLU}(\ell_j - m + 1), \qquad u_j = \mathrm{ReLU}\left(e_j + \mathrm{ReLU}\left(1 - \sum_{k<j} e_k\right) - 1\right).$$

The e_j indicate maximizing logits and u is one-hot at the first maximizer. All prefix sums use one dense affine map, so width remains $O(V)$. Multiplication by the source embedding table recovers the next embedding with the source's prescribed position-combination arithmetic.

Let $a' = a + 1$. The exact write flag at integer address i is

$$b_i = \mathrm{ReLU}(1 - \mathrm{ReLU}(i - a') - \mathrm{ReLU}(a' - i)).$$

For $|v| \leq U$ and binary b, the constant-depth gate

$$G(v, b) = \mathrm{ReLU}(v + Ub) - \mathrm{ReLU}(v - Ub) - Ub = bv$$

implements a conditional write as $G(v_{\text{new}}, b) + G(v_{\text{old}}, 1-b)$. Apply it to embeddings and indicators, increment every cursor copy, and reset scratch channels. Only row $a+1$ changes its token. Induction proves the claimed trajectory; the final token is at known address N. The complete trace, if requested, incurs its output cost.

Control integers have magnitude $O(2^p + N^2 + V)$, so the stated p' suffices. Widening control registers does not alter copied source quantizers. A fixed stack of K layers performs every iteration; weights within that stack need not coincide. Each dense layer costs $O(NW^2 + N^2W)$ operations, including padded rows, all vocabulary heads, and zero blocks in dense matrices. Reusing layer buffers gives the data memory bound. Dense matrices cost $O(KW^2p')$ bits; source embedding data and $O(N)$ address fields give the remaining description terms. Constructing addresses costs $O(N)$ word operations; initializing all registers costs $O(NW)$ writes plus source encoding. These are charged preparation operations. All work bounds concern declared word primitives; bit-work bounds require their digital implementation costs. □

A.2 Loop to CoT

Represent the source loop by an S-bit state, admitted initialization and Boolean readout circuits of G_0, G_{out} NAND gates, and a shared transition circuit of G NAND gates. Constants and wire fan-out are allowed, and the transition may read the input. These circuits implement the source architecture's operations; they grant no extra preprocessing. State-output wire lists and all circuit descriptions are charged.

Lemma 10 (Circuit compilation into tokens)**.** *For horizon $R \geq 1$, put $K = G_0 + RG + G_{\text{out}} + 2$ and $N = n + 2 + K$. A causal normalization-free hard-attention/ReLU decoder of constant depth and width, with two heads and charged absolute positional embeddings, emits at most K binary tokens whose last token equals the source answer. Exact signed precision $2\lceil \log_2 N \rceil + O(1)$ suffices. Parameters and data workspace each occupy $O(N \log N)$ bits. Including prefill, inference work is $O(N^2)$ with dense attention and $O(N)$ under certified positional gathers.*

If D_{src} is the circuit description length in words, including state-output lists, compilation takes $O(D_{\text{src}} + N + RS)$ address operations and $O(S + N)$ temporary words in addition to the source description. These words have $O(\log(n + S + G_0 + G + G_{\text{out}} + R + 2))$ bits. Compilation is added whenever preparation is included in the budget. The decoder retains no additional mutable S-bit state.

Proof. Unroll initialization, R transition copies, and readout into a topologically ordered NAND circuit. Append two self-NAND gates to copy the designated answer to the last gate, even if the readout originally named an input wire. Prefix input tokens by constants $0, 1$ at positions $0, 1$, and let $n_0 = n + 2$. Gate k writes position $n_0 + k - 1$; its operand addresses $a_k, b_k \leq n_0 + k - 2$ are stored in the positional embedding at the preceding position, which predicts gate k. Unused prefill positions request address zero.

Protected residual channels retain the token bit, j, j^2, two operand addresses, and a constant one; scratch channels start at zero. Two heads with keys (j, j^2) and respective queries $(2a_k, -1)$, $(2b_k, -1)$ select the required bits from the self-inclusive causal prefix by the same quadratic score identity used above. Their values read the protected original token bit, not an earlier computed prediction. The feedforward map computes

$$h = 1 - \text{ReLU}(u + v - 1) = \text{NAND}(u, v),$$

and logits $(1 - h, h)$ emit its value without ties. Induction on the gate order proves correctness. One attention–feedforward block suffices, up to constant layer overhead for head-layout conventions. The same weights evaluate every gate; only input-independent wiring instructions in the charged positional table vary.

All scores and intermediates have magnitude $O(N^2)$, proving the precision bound. The positional table has $O(N)$ scalar entries and other parameters have constant size. Tokens, KV records, and streaming attention reductions take $O(N \log N)$ data bits. Dense work is

$$O\left(n_0^2 + \sum_{k=0}^{K-1} (n_0 + k)\right) = O(N^2).$$

Under the gather evaluator, both heads directly address the complete causal interval; all prefill and decoding computations cost $O(n_0 + K)$. Realizing addresses and squares adds $O(N)$ work, while streaming maxima avoid a quadratic score array. No input-dependent mask is used.

To compile, maintain an S-entry map from state wires to constants, input positions, or gate-output positions. Translate initialization, then each transition's gates and state-output list using old and new maps. Output-wire aliases require no generated token. Translate readout and append the two copy gates. Reading the source and writing gates costs $O(D_{\mathrm{src}} + N)$ operations; updating maps costs $O(RS)$; two maps and the instruction list take $O(S + N)$ temporary words. The stated compiler word width covers identifiers, counters, and emitted addresses. This preparation cost is separate from inference and cannot be bounded by the token count alone when many state wires are aliases. Its peak memory is checked as a separate phase whenever preparation is budgeted. □

The circuit size G is a computational quantity, not just the number of latent bits. For ordinary digital primitives it has an explicit polynomial bound:

Proposition 11 (Digital block circuit size)**.** *Consider L layers, m mutable and n_f frozen source rows, width d, a constant number of heads, and feedforward width $O(d)$. Suppose numerical intermediates use $O(p)$ bits and each arithmetic, ReLU, comparison, and quantization primitive has an $O(p^2)$-gate Boolean circuit; ties follow fixed source order. If frozen per-layer keys and values are prepared by the admitted initializer and stored, then*

$$G = O\big(G_{\mathrm{mask}} + L[md^2 + m(m + n_f)d]p^2\big)\,,$$

where G_{mask} charges nonconstant-mask generation. An affine binary readout costs $O(dp^2)$ gates. Frozen-cache construction enters G_0 and its $O(Ln_f dp)$ bits enter memory. Reconstructing frozen representations each call instead adds their encoding cost G_x and $O(Ln_f d^2p^2)$ projection gates to G.

Proof. Mutable projections and feedforward maps use $O(Lmd^2)$ scalar operations; scores use $O(Lm(m + n_f)d)$. Score/priority comparisons and value selection are absorbed by the stated bound. Compile the stipulated primitives and convert to NAND with constant overhead. Stored frozen caches have identity output-wire aliases, costing storage but no transition gates. Reconstructing them yields the additional terms. Ordinary schoolbook arithmetic satisfies the primitive hypothesis; arbitrary lookup primitives do not acquire that complexity by assumption. □

B Lower bounds for tokens, loop calls, and work

All constants below may depend on the prescribed architecture, precision, initialization, and controller caps and on $\delta < 1/2$, but not on n, m, κ, or the input. Statements specialized to a constant imbalance cap κ_0 may also depend on that fixed constant. Expectations include independent post-prefill coins. The assumptions of Section 3, including local control and charged input access, apply throughout.

B.1 A common query argument for contextual inputs

We use the simultaneous restriction lemma of Amiri et al. (2025, Lemma 3.4, restated as Lemma B.4). For the deterministic, bounded-depth and bounded-head prefill, one restriction ρ leaves $N \geq n/2$ raw bits free and makes *every* activation, at every position and layer, a function of at most a constant c_0 free coordinates. This holds for causal and bidirectional input attention, including pointwise maps. A frozen or initial mutable record combines only a bounded number of activations at fixed positions and layers; it therefore depends on at most c free bits, for a fixed $c \geq 1$. The same restriction covers all these records. Their support sets may overlap. We do not assume that contextual records were input-local before restriction, or expose the union of all their supports.

A fixed number of layout symbols causes no change: start the prior-restriction form of the cited Lemma B.3 with those positions fixed and retain half the enlarged input length. For sufficiently large n, at least $n/2$ raw coordinates remain free. A layout symbol can use a fixed raw embedding plus a position-specific constant offset. The restriction depends only on deterministic prefill, not on later coins. On its free coordinates, parity is parity of N independent bits, possibly complemented.

Lemma 12 (Simulating a contextual hard-attention query)**.** *Suppose the mutable state and query-side controls are known, and each frozen key/value record depends on at most c still-unrestricted raw coordinates. Under a uniform completion of the current raw-bit transcript, one head can be simulated exactly using at most $c2^c$ new coordinate queries in conditional expectation. Alternatively, fixing at most c additional coordinates suffices to make that head's output constant on a nonempty subcube.*

Proof. Order attainable candidate outcomes by score and the fixed source tie priority. Enumerate compatible assignments to each frozen candidate's support and choose a greatest possible outcome. Mutable and constant sources are already known. Fixing the selected candidate's support to an attaining assignment makes it win on every remaining completion: no other candidate can attain a greater outcome. Overlapping supports only remove possibilities. This proves the certificate assertion.

For the query simulation, reveal that candidate's remaining support instead of fixing it. If $k \le c$ coordinates are unknown, they attain the chosen assignment with conditional probability $2^{-k} \ge 2^{-c}$; then the exact winner and value are known. Otherwise recompute the greatest possible outcome and repeat. Unrevealed bits remain independent and uniform, since choices use only the explicit reveal transcript and independent coins. Each failed attempt exposes at least one new bit. The expected number of reveals is at most $c\sum_{j\ge 0}(1-2^{-c})^j = c2^c$. Enumeration is a query-complexity simulation, not an efficient arithmetic implementation. □

Theorem 13 (Contextual input does not remove update volume)**.** *Let Q count attention queries whose source banks include frozen input records. Exact parity requires $Q + m \ge n/(2c)$ on some input. For almost surely terminating randomized loops with per-input error at most $\delta < 1/2$,*

$$\sup_x \mathbb{E}Q(x) \ge \frac{(1-2\delta)n/2 - cm}{c2^c}. \tag{2}$$

In particular, $\bar{R} = \Omega(n/m)$ when $m = o(n)$, since $Q \le HLmR$.

Proof. On ρ, first expose the union of initializer supports, of size $b \le cm$. The entire mutable state is then known. Simulate heads chronologically with Lemma 12; after each layer, the exact next state, control choices, and stopping decision are known. Queries within a layer may be simulated in any fixed order: their query vectors still use the preceding-layer state. If K is the number of distinct coordinates revealed, conditional expectation gives

$$\mathbb{E}K \le b + c2^c\mathbb{E}Q.$$

For input-dependent stopping, sum the costs with indicators of executed queries and apply Tonelli's theorem. Infinite expected cost makes the claim immediate. The simulator preserves the output on each input and terminating tape.

A final transcript with $K < N$ leaves parity unbiased, while the answer is fixed by that transcript and the tape. Thus $\Pr(K = N) \ge 1 - 2\delta$ and $\mathbb{E}K \ge (1-2\delta)N$. This proves (2), since an input supremum dominates the average on ρ. For exact computation, instead apply the certificate part of the lemma after fixing initializer supports. Halting with $b + cQ < N$ would give a constant answer on a subcube where parity is nonconstant. Nested nonempty restrictions give the asserted worst-case input. □

B.2 Sampled CoT with its complete contextual cache

Lower bound in Theorem 4. Apply the same simultaneous restriction to deterministic causal prefill. Each cached input key or value is a pointwise projection of a prefill activation and hence depends on at most c free coordinates. Prefix consistency keeps those functions unchanged during generation. The prefill state that determines the first next-token distribution also has support at most c; reveal it first, and couple all sampling and stopping to the decoder's independent tape.

Induct over generated positions and decoder layers. Earlier generated tokens and all their per-layer KV records are known exactly. The next token's embedding is known after sampling. At each layer, its query is known; its sources are the c-local frozen input cache and known generated-prefix records, including the current position if permitted. Lemma 12 computes each selected value exactly. Execute the actual pointwise map, retain the actual KV record, and continue to the next layer. The resulting

next-token distribution, local stopping rule, and token-based answer are exactly preserved. There is no assumed feedback between cached decoder layers.

For T generated tokens, including the answer, there are at most $L_C H_C T$ newly processed head evaluations. Therefore

$$\mathbb{E}_{\rho,\omega} K \le c + c2^c L_C H_C \mathbb{E}_{\rho,\omega} T.$$

The same executed-query/Tonelli argument handles adaptive stopping. The unread-bit argument gives $\mathbb{E}K \ge (1-2\delta)n/2$, and consequently

$$\sup_x \mathbb{E}T(x) \ge \frac{(1-2\delta)n/2 - c}{c2^c L_C H_C} = \Omega(n).$$

The exact running-parity decoder in Appendix C gives the matching upper bound. Neither this argument nor Theorem 13 asserts a uniform-average lower bound on the original cube: the hard subcube can have small original measure. Deterministic bounded-depth prefill and the per-input error guarantee are essential. □

B.3 The joint work–call lower bound

Flatten loop layers into stages. With the setup's executed row charges, write $q_s = \max_i a_{si}$, $J \le LR$, and $W = \sum_{s=1}^{J} q_s$; let $d \ge 1$ be the fixed controller-count cap. Temporarily permit any imbalance cap $\kappa \ge 1$; all constants in the following joint inequality are independent of κ. Every row executes, and the cap gives

$$C_{\text{update}} \ge a_0 \sum_{s,i} a_{si} \ge \frac{a_0 m}{\kappa} W. \tag{3}$$

This counts masked and repeated scan candidates and unused phase heads; additional descriptor or arithmetic work only increases the charge.

On ρ, fix the independent tape. Construct an adaptive raw-query simulator that first computes the at most d designated initial controller/readout records, then computes and retains their histories stage by stage. To recursively compute any stage-s row, first compute its own stage-$(s-1)$ record. Together with the known controller history, this determines its actual kernel, bank cardinalities and iterators, gather addresses, and query-side parameters. For a gather compute its selected source; for a scan compute every listed source. Mutable sources are from stage $s-1$; frozen sources terminate at their at most c raw coordinates. Execute the exact attention and row update.

The recurrence needs at most $c\prod_{j\le s} q_j$ raw queries for a stage-s row, conditional on known designated histories. Iterators and membership use no unselected source contents. The q_j are the maxima of the *actual* execution and bound branching retrospectively; the simulator need not discover them by inspecting unrelated rows. Repeated queries or memoization only reduce the bound. Including all designated histories, the distinct reveal count obeys, pathwise,

$$K \le cd\left(1 + \sum_{s=1}^{J} \prod_{j\le s} q_j\right) \le cd(J+1)\prod_{s=1}^{J} q_s. \tag{4}$$

For $J = 0$, this is $K \le cd$. Local halting ensures this simulates the actual stopping decision as well as the answer.

Under uniform free bits and independent tapes, correctness requires $\Pr(K = N) \ge 1 - 2\delta$. For sufficiently large n, this event has $J \ge 1$. AM–GM and $J + 1 \le 2^J$ give on this event

$$\log(N/(cd)) \le J\log(2W/J) \le F(J,W), \qquad F(u,v) = u\log(1 + 2v/u).$$

Define $F(0,0) = 0$. This perspective is jointly concave, nonnegative, increasing in each argument, and bounded above by $2v$. For finite expected work, Jensen's inequality and (3) therefore imply

$$L\bar{R}\log\left(1 + \frac{2\kappa\bar{C}}{a_0 m L\bar{R}}\right) \ge (1-2\delta)\log(N/(cd)). \tag{5}$$

For fixed $\kappa = \kappa_0$ this gives the joint lower bound in Theorem 5; its explicit form also applies to the varying cap in Section 6. It uses neither a fixed stopping horizon nor a common worst-case input for

work and calls. In particular, $F \le 2W$ gives $\bar{C} = \Omega(m \log n/\kappa)$, with an absolute dependence only on the other fixed model parameters.

There is an independent $\Omega(n)$ work floor for both architectures. Simulate the entire charged computation, including prefill, by revealing a raw input record when a primitive first accesses it. A word primitive accesses only constantly many separately addressed raw records. On uniform raw inputs the unread-bit argument forces at least $(1-2\delta)n$ expected distinct reads. Hence $\bar{C} = \Omega(n)$, and loop work is $\Omega(n + m \log n/\kappa)$. At fixed κ_0 this is $\Omega(n + m \log n)$.

We also obtain $m\bar{R} = \Omega(n)$ for the full range $1 \le m \le n$. For $m \le (1-2\delta)n/(4c)$, use (2) and $Q \le HLmR$. For larger m, the full-reveal event in (4) requires a call and has probability at least $1-2\delta$, so $\bar{R} \ge 1-2\delta$.

Matching work and capacity consequences. In this paragraph specialize to the fixed cap $\kappa = \kappa_0$. Appendix C constructs exact parity with $R = O(n/m + \log m)$ calls, work $O(n + m \log n)$, and $O(n \log n)$ stored and program bits; its CoT construction has $T = \Theta(n)$ and $O(n)$ work. Thus the work lower bounds match. At loop work $\bar{C} \le K(n + m \log n)$ for a fixed K, if $m \le n/\log n$, volume gives $\bar{R} = \Omega(n/m) \ge \Omega(\log n)$. Otherwise $\bar{C} \le 2Km \log n$; writing $u = \bar{R}/\log n$, (5) rules out $u \to 0$ because $u \log(1 + O(1)/u) \to 0$. Therefore the fewest calls at the optimal work order are $\Theta(n/m + \log n)$. Under a linear work budget, this requires $m = O(n/\log n)$. With an available capacity of at most M_{row} rows, choose a power of two within a constant factor of $\min\{M_{\mathrm{row}}, n/\log n\}$ in the construction. The resulting call order is $\Theta(n/M_{\mathrm{row}} + \log n)$, proving Corollary 6 for a sufficiently large fixed linear-work multiplier.

B.4 A CALL FLOOR FOR UNRESTRICTED NUMERICAL RECORDS

Theorem 14 (Finite-record call depth)**.** *Under the setup's fixed caps and $O(\log n)$-bit whole records, any per-input bounded-error parity loop has*

$$\bar{R} = \Omega(n/m + g_n), \qquad g_n = \log n/\log\log n.$$

This statement imposes no restriction on the number of possible numerical values retained by a row, and needs no work-imbalance bound.

Proof. Apply the common prefill restriction. Each initial record bit is a Boolean function of at most c free coordinates, hence has a constant-depth, polynomial-size representation. Fix a tape. A function of constantly many $O(\log n)$-bit records has a polynomial-size, depth-two DNF, regardless of its charged numerical implementation.

For attention at destination i, let E_{ijk} assert that j is eligible and k is ineligible or outranked by j, including tie priority. For $j = k$, use eligibility. This predicate reads records i, j, k and the fixed number of preceding controller records. Local bank membership, kernel/certificate selection, scores, and masks add no record arguments. Thus E_{ijk} has a polynomial-size constant-depth representation. With $O(n)$ records, write

$$w_{ij} = \bigwedge_k E_{ijk}, \qquad o_{i,b} = \bigvee_j (w_{ij} \wedge v_{j,b}).$$

The projected value bit $v_{j,b}$ and subsequent pointwise update have the same finite-record representation; an empty bank uses its fixed default. The polynomially many triples give constant depth and polynomial size per layer. Carry complemented signals in parallel to place negations at inputs. Unrolling r calls gives fixed constants A, K, d_0, d_1 such that

$$S \le A(r+1)n^K, \qquad d \le d_0 + d_1 r. \tag{6}$$

Metadata and coins are hardwired. This circuit represents the declared computation; it grants no new inference primitive or free global control.

For average sensitivity $\mathrm{as}(g)$, the depth-uniform bound of Rossman (2018, Theorem 4 and Fact 5) implies

$$\mathrm{as}(g) \le (C \log(2S))^{d-1} \tag{7}$$

with an absolute C, padding to depth at least two if necessary. Indeed, unfold a size-S, depth-d circuit into a formula of at most S^{d-1} gates and apply the cited gate-count bound; its $(d-1)^{-1}$

factor cancels. Uniformity as d grows is essential. If g has average parity error $\eta < 1/2$, then for every coordinate, both endpoints of a uniform edge are correct with probability at least $1 - 2\eta$. Consequently $\mathrm{as}(g) \geq (1 - 2\eta)N$. For $r \leq n$, (6) has $\log(2S) = O(\log n)$; combining these bounds gives $r = \Omega(g_n)$. For $r > n$ the same lower bound is immediate.

For expected stopping, put $\alpha = (1/2 - \delta)/2$ and truncate at $r = \max\{1, \lceil \bar{R}/\alpha \rceil\}$ calls, returning a fixed bit on timeout. Markov's inequality raises average error on ρ by at most α. Compile local halt/readout at each stage, retaining the first halted answer with a Boolean accumulator. This adds constant depth per stage and preserves (6); syntactic continuation after halt does not alter the answer. Fix a tape of average error at most $\eta = \delta + \alpha < 1/2$. The preceding argument gives $r = \Omega(g_n)$ and hence $\bar{R} = \Omega(g_n)$. Combine this with (2) when $m \leq (1 - 2\delta)n/(4c)$. For larger m, $n/m = O(1)$ is already dominated by g_n. This proves the statement. □

C Exact parity implementations

We give the linear-work upper bounds used in Theorem 5. All operations below use constant-width hard-attention/ReLU blocks, exact $O(\log n)$-bit signed arithmetic, and the certified positional gathers of Section 3.

A stationary binary loop. First let the number of logical rows k be a power of two. Put $b = \lceil n/k \rceil$ and append fixed zero input rows to length $n' = kb < n + k$. Frozen row j stores x_j, j, j^2. Logical row $i \in \{0, \ldots, k-1\}$ retains its position, a Boolean accumulator a_i, loading address ℓ_i, clock t, and common offset q, initialized by

$$a_i = 0, \quad \ell_i = i + 1, \quad t = 0, \quad q = 1.$$

For integer u, let $s(u) = \mathrm{ReLU}(u + 1) - \mathrm{ReLU}(u) = \mathbf{1}[u \geq 0]$. The phase and partner address are

$$\pi = s(t - b), \qquad j_i = i + q - ks(i + q - k).$$

During reduction $q = 1, 2, \ldots, k/2$, so $j_i = (i + q) \bmod k$. For $k = 1$ there is no reduction and the unused partner address is zero. A frozen-input head with query $(2\ell_i, -1)$ and key (j, j^2) gathers $v_i = x_{\ell_i}$ during loading. A mutable head with the same quadratic score gathers $w_i = a_{j_i}$ from the preceding state. Both heads execute in both phases; addresses outside a frozen interval are clamped.

Use the exact Boolean XOR map and bounded gate

$$X(u, v) = u + v - 2\,\mathrm{ReLU}(u + v - 1), \qquad G_U(c, u) = \mathrm{ReLU}(u) - \mathrm{ReLU}(u - Uc) = cu$$

for binary u, v, c in the first expression and $c \in \{0, 1\}$, $0 \leq u \leq U$ in the second. One shared block performs

$$a_i' = G_1(1 - \pi, X(a_i, v_i)) + G_1(\pi, X(a_i, w_i)),$$
$$\ell_i' = \ell_i + (1 - \pi)k, \qquad q' = q + G_k(\pi, q), \qquad t' = t + 1.$$

Protected residual channels preserve positions and constants. Preparatory maps, signed scratch channels, identity attention, and scratch resets use only a fixed number of sublayers and are charged. These sublayers form one block shared across all calls.

After b loading calls, $a_i = A_i := \bigoplus_{u=0}^{b-1} x_{i+1+uk}$. After r subsequent reduction calls, induction gives $a_i = \bigoplus_{u=0}^{2^r-1} A_{(i+u) \bmod k}$. The two intervals joined at each step are disjoint until they cover all k rows. Thus row zero returns parity after $b + \log_2 k$ calls.

For an arbitrary physical workspace m, take the greatest power of two $k \leq m$. Every destination uses the same fixed first-k mutable bank. Each padding row duplicates logical row zero's fields and computation; padding rows never enter that source bank. They still execute every head and local map. Because $k > m/2$, every call costs $O(m)$ work and

$$R = O(n/m + \log m), \qquad C = O(n + m \log n).$$

All cursors and squared-address scores have polynomial magnitude in n; the unused loading address is at most $n' + k$. Positions, frozen bits, double-buffered states, streaming scratch, and parameters occupy $O(n \log n)$ bits. Their initialization takes $O(n)$ word work. Input-dependent data are Boolean at call boundaries; positions and clocks are input independent there, so the construction satisfies a fixed retained-data alphabet.

A causal running-parity decoder. Append an input-independent delimiter representing $z_0 = 0$. At the position predicting token z_k, a fixed positional map supplies address k; a quadratic positional gather retrieves raw input x_k. The current token supplies z_{k-1}, and the map X emits $z_k = z_{k-1} \oplus x_k$. The address lies in the raw input, which is causally available. The state is the ordinary token and KV cache, with no separate accumulator buffer. One layer and one head suffice. The prefill keys and values are projections of input/position fields; its attention also uses a certified clamped gather. Consequently prefill and the n token steps together use $O(n)$ word work and $O(n \log n)$ stored/program bits. The final token is exact parity.

D MATCHING CONSTRUCTION FOR THE BUDGET FRONTIER

The construction uses increasing fan-in parity modules, then caps their fan-in to control scan work. This embeds a classical truth-table reduction in one stationary Transformer block (Beame & Hastad, 1989). All physical rows and all declared heads execute on every call.

Theorem 15 (Capped parity reduction)**.** *Let $64 \le m \le n$, set $N_0 = \lfloor m/32 \rfloor$, and let r_* be the least positive integer with $(r_* + 1)! \ge N_0$. For every integer $2 \le b \le r_* + 1$, an exact parity loop with exactly m evaluated rows satisfies*

$$R = O\left(\frac{n}{m} + \frac{\log m}{\log b}\right), \qquad C = O\left(n + \left(n + \frac{m \log m}{\log b}\right) 2^b\right).$$

Its width, block depth, and head count are constant, its precision is $O(\log n)$, and its total stored and program bits are $O(n \log n)$. It has a fixed constant data alphabet at completed-call boundaries, apart from input-independent metadata and the permitted controllers. Bank membership is fixed by destination; scan cardinalities and kernel types are shared across destinations at each stage. Constants are independent of n, m, b.

Proof.

A constant-width parity module. For a group of k bits $v_1, \ldots, v_k$, allocate one literal helper for each odd pattern $a \in \{0,1\}^k$, and one aggregator. A helper's two fixed-mask hard-attention heads return

$$u_a = \max(\{0\} \cup \{v_i : a_i = 0\}), \qquad w_a = \min(\{1\} \cup \{v_i : a_i = 1\}).$$

Scores v_i and $-v_i$ implement these extrema. The constant-size ReLU map $z_a = \mathrm{ReLU}(w_a - u_a)$ is exactly the pattern's match indicator. The aggregator returns $\max_a z_a$, the parity bit. Ties have equal values and follow the fixed source order. Each helper has its preceding group and two constant sentinels in its physical bank. For an incomplete group, missing inputs mean zero: a pattern requiring a missing one admits the zero sentinel in its minimum head, making that indicator zero. Missing zero literals need no source. Pattern bits are charged fixed mask entries; no numerical bit-extraction operation or growing query vector is used.

A fixed reduction graph. Allocate N_0 base accumulators. Above them form consecutive groups with successive arities $2, 3, \ldots, b$, stopping early if only one value remains. At prefix level j, there are $N_j = \lceil N_0/(j+1)! \rceil$ modules, each using 2^j helpers and an aggregator. If the prefix ends with more than one output, their count is $U = \lceil N_0/b! \rceil$. Repeatedly group these output references in batches of at most b. A batch of actual size $k \ge 2$ uses the module above; carry a singleton by referring to its existing output. Resolve these aliases in the fixed source lists, without extra mutable rows or inference operations.

The tail has at most $\lceil \log_b U \rceil$ levels. If h is the maximum number of modules on any base-to-root path, then $h \le b - 1 + \lceil \log_b \lceil N_0/b! \rceil \rceil$ when a tail exists, and $h \le b - 1$ otherwise. The admitted range implies $(b-1)! < N_0$. Elementary factorial bounds give $b \log b = O(\log N_0)$ uniformly, absorbing bounded b into the constant. Thus $h = O(\log m / \log b)$.

One stationary shared block. Mutable values start at zero. Base i, $0 \le i < N_0$, maintains a cursor $p_i = i$ and accumulator $a_i = 0$. Every call performs three fixed stages on every row:

1. Refresh each base by gathering the raw bit at its clamped cursor, setting $a_i \leftarrow a_i \oplus (\mathbf{1}[p_i < n] x_{p_i})$, and incrementing p_i by N_0.

2. Refresh every helper from the just-refreshed bases and the preceding-call values of its aggregator children.
3. Refresh every aggregator from its just-refreshed helpers.

The input gate discards the clamped bit outside the input. Certified positional scores $2p_i j - j^2$ give the input gather. Boolean XOR, thresholds, role gates, and protected-register preservation are the constant-size ReLU operations in Appendix C. Fixed role fields preserve registers whose outputs are unused, while their heads still execute. The block has constant architecture independently of the graph, arity, and cap; it uses no variable multiplication or changing external program.

After $B_0 = \lceil n/N_0 \rceil$ calls, the bases hold the parities of disjoint strided groups covering the input and remain unchanged. Once its children are permanently correct, a module becomes permanently correct within one further call. Induction on graph height, also for children at different heights after singleton aliasing, makes the root correct after $B_0 + h$ calls. A charged counter selects this fixed horizon. Higher modules execute during loading; their transient values need not yet be correct. Data heads retain bits, not winner indices; temporary numerical scratch is cleared by the constant-size role scaffolding. All cursors, clocks, and roles are input-independent at a fixed boundary. Thus each ordinary complete record has a fixed finite retained-data alphabet.

Rows and description size. For a factorial prefix of $r \le r_*$ levels, $r! < N_0$. Its row count, including bases, is at most

$$N_0 \left[1 + \sum_{j \ge 1} \frac{2^j + 1}{(j+1)!} \right] + (2^{r+1} - 2) + r < 9N_0.$$

Indeed the bracket is $1 + (e^2 - 3)/2 + (e - 2) < 4$, $2^r \le 2r!$, and $r < N_0$. A tail is a tree on U prefix outputs with at least two children at every genuine internal node, hence at most $U - 1$ modules. Its rows number at most

$$(2^{b-1} + 1)(U - 1) < N_0 \frac{2^{b-1} + 1}{b!} \le \tfrac{3}{2} N_0.$$

Fewer than $11N_0 < m$ mutable rows suffice; pad to exactly m and evaluate the padding as well.

The prefix's mask and adjacency incidence is

$$O\left(\sum_{j=1}^{r} N_j j 2^j \right) = O\left(N_0 \sum_{j \ge 1} \frac{j 2^j}{(j+1)!} + r 2^r \right) = O(N_0),$$

using $r2^r \le 4r!$. The tail needs at most $O(N_0 b 2^b / b!) = O(N_0)$ entries, since $b2^b/b! \le 4$. Patterns and entries can be enumerated in their output-linear work; geometrically decreasing reference lists resolve aliases during layout construction. No factorial arithmetic is needed during inference. Construction therefore costs $O(m)$ words and operations beyond the $O(n)$ input construction.

Shared scan costs and precision. Give both literal heads common cardinality $q_H = b + 2$ and the aggregator head cardinality $q_G = 2^{b-1}$. Append masked frozen dummy slots to each destination's live list. An unused role has an eligible zero sentinel and ignores the result, while still paying the full scan. Input and identity heads use shared certified gathers. Store a live-list pointer, length, and dummy-interval descriptor rather than materializing every padded list. A streaming scan enumerates and charges every live or dummy slot with constant word work per slot. A shared pool of $O(b + 2^b) = O(m)$ distinct frozen constants suffices, because $2^{b-1} \le 2(b-1)! < 2N_0$. Dummy IDs follow live IDs without changing any selected value or tie rule. Pool and descriptor construction cost $O(m)$ words and work.

Each call costs $O(m(1 + 2q_H + q_G)) = O(m2^b)$, including unused heads, padding, and identity scaffolding. Since $mB_0 = O(n)$, this gives

$$C = O(n + m(B_0 + h)2^b) = O\left(n + \left(n + \frac{m \log m}{\log b} \right) 2^b \right).$$

Raw input, mutable rows, frozen constants, live lists, masks, descriptors, counters, and streaming state use $O(n)$ words. Addresses, cursors, and positional-score intermediates have polynomial magnitude,

so one fixed $O(\log n)$ precision coefficient suffices. This also charges all resident and compiled program information. □

Lemma 16 (Partial padding with a prescribed imbalance cap)**.** *Under the parameter assumptions of Theorem 15, let $\kappa \geq 1$ be an integer. There is an exact parity loop with exactly m evaluated rows and stagewise imbalance at most κ, satisfying*

$$R = O\left(\frac{n}{m} + \frac{\log m}{\log b}\right),$$
$$C = O\left(n + \left(n + \frac{m \log m}{\log b}\right)\left(1 + \frac{2^b}{\kappa}\right)\right).$$

Its architecture is constant, its precision is $O(\log n)$, and its stored and program bits are $O(n \log n)$. Its completed-call data alphabet is fixed independently of n, m, b, κ. Every row, head, listed source, and masked slot is evaluated. All constants are uniform in these parameters.

Proof. Keep the logical graph and shared update of Theorem 15. Set

$$q_H = b + 2, \qquad q_G = 2^{b-1}, \qquad F_H = \lceil q_H/\kappa \rceil, \qquad F_G = \lceil q_G/\kappa \rceil.$$

These positive integer floors depend only on the prepared layout and the prescribed cap. For a literal or aggregator bank with actual live length ℓ, give it physical cardinality $\max\{\ell, F_H\}$ or $\max\{\ell, F_G\}$, respectively, using masked dummy slots after the live list. Literal live lists include both sentinels even when one is ineligible. Unused roles have eligible zero singleton lists and still execute their declared scans. No source, winner, or local-map output changes. In particular, the call schedule and retained data of the earlier construction are unchanged.

Balance at every stage. At a literal stage write c_H for the fixed per-row baseline plus its preservation gathers. Its two scan heads have live lengths at most q_H, and both are padded to at least F_H. Thus

$$c_H + 2q_H/\kappa \leq \min_i a_{Hi} \leq \max_i a_{Hi} \leq c_H + 2q_H.$$

The aggregator stage similarly satisfies $c_G + q_G/\kappa \leq \min_i a_{Gi} \leq \max_i a_{Gi} \leq c_G + q_G$. For either stage, writing $h = 2$ or 1 gives

$$\frac{m \max_i a_{si}}{\sum_i a_{si}} \leq \frac{c_s + hq_s}{c_s + hq_s/\kappa} \leq \kappa.$$

Loading, preservation-only, and fixed local-map stages execute equal constant-cost kernels on every row. They therefore also satisfy the cap. These inequalities use only the fixed layout; no runtime maximum or global scheduling operation is supplied to a row.

Live incidence of the entire capped graph. At prefix level j, each of its N_j modules contributes at most $2^j(2j+7)$ scan slots: two child-and-sentinel lists per literal helper, followed by the aggregator's helper list. If the prefix has r levels, then

$$\sum_{j=1}^{r} N_j 2^j (2j+7) \leq N_0 \sum_{j \geq 1} \frac{2^j(2j+7)}{(j+1)!} + O(r2^r) = O(N_0).$$

Here $r! < N_0$ and $r2^r \leq 4r!$. The capped tail has at most $U - 1 < N_0/b!$ genuine modules. A module of actual arity $k \leq b$ contributes $2^{k-1}(2k+5) = O(b2^b)$ live scan slots. Since $b2^b/b! \leq 4$, its entire tail contributes $O(N_0)$ slots. Singleton aliases add no module or mutable row. Unused-role singleton scans add at most $3m$ slots, and all loading/preservation gathers contribute $O(m)$. Thus total live incidence per call is $O(m)$, uniformly in the cap, including the tail and every unused role.

Partial padding adds at most $2mF_H + mF_G$ slots. Since $F_s \leq 1 + q_s/\kappa$ and $b + 2 \leq 2^b$ for $b \geq 2$, the full-call work is

$$O\big(m(1 + F_H + F_G)\big) = O\left(m\left(1 + \frac{2^b}{\kappa}\right)\right).$$

The previous bound $R = O(n/m + \log m/\log b)$ and charged $O(n)$ preparation give the claimed total work.

Preparation, storage, and precision. Each bank stores its original live-list pointer and length plus a constant-size dummy-interval descriptor. The live-list and mask count just proved is $O(m)$. A shared pool of $O(q_H + q_G) = O(m)$ frozen dummy records suffices, because $2^{b-1} \le 2(b-1)! < 2N_0$ and $b + 2 = O(N_0)$. Constructing the pool, enumerating and validating the live entries, and assigning the per-row floor descriptors takes $O(m)$ word work and words. The floors may be computed by integer quotient and remainder during preparation; no division or ceiling operation is required inside the shared block. Equivalently, at most $O(m)$ repeated subtractions compute both floors. Only $\min\{\kappa, \max(q_H, q_G)\}$ is needed to specify them; larger caps do not enlarge the compiled representation. The expanded padded lists are never stored.

Together with raw input, counters, buffers, local-map scratch, and resident program information, this uses $O(n)$ words. All bank sizes, addresses, cursors, and scores retain polynomial magnitude, independently of κ. Masked dummy slots follow live slots, so eligibility and tie priority are unchanged. The same fixed precision coefficient and retained-data alphabet therefore suffice. At $\kappa = 1$ the banks have the common cardinalities of Theorem 15; when $\kappa \ge \max(q_H, q_G)$ there is no dummy padding. Both endpoints retain all-row and all-head execution. □

Corollary 17 (A uniform work–capacity–imbalance upper bound)**.** *There are fixed constants K_0, C_0 such that, for integer $1 \le \kappa \le M_{\mathrm{row}} \le n$ and $B \ge K_0 n$, an exact loop with a chosen workspace $m \le M_{\mathrm{row}}$, work at most B, and stagewise imbalance at most κ uses*

$$R \le C_0 \left(\frac{n}{M_{\mathrm{row}}} + \max\left\{ g_n, \frac{\log n}{\log(2 + \log(2 + \kappa B/n))} \right\} \right).$$

For sufficiently large n, $g_n = \log n / \log\log n$; bounded input lengths are handled directly. The architecture, whole-record precision coefficient, and constant retained-data level are fixed uniformly in $n, B, M_{\mathrm{row}}, \kappa$, and stored/program bits are $O(n \log n)$.

Proof. If $M_{\mathrm{row}} \le n/\log n$, use the binary construction of Appendix C with $m = M_{\mathrm{row}}$. It has equal row costs, $O(n)$ work, and $O(n/M_{\mathrm{row}} + \log M_{\mathrm{row}}) = O(n/M_{\mathrm{row}})$ calls. This includes every small integer capacity. Hence assume $M_{\mathrm{row}} > n/\log n$ and n sufficiently large.

Choose a fixed K large enough for the construction constants below, and $K_0 \ge 4K$ large enough also for the binary fallback. Put

$$b = \min\left\{ \left\lfloor \log_2 \frac{\kappa B}{Kn} \right\rfloor, \left\lfloor \frac{g_n}{4} \right\rfloor \right\}, \qquad m = \min\left\{ M_{\mathrm{row}}, \left\lfloor \frac{n \log b}{\log n} \right\rfloor \right\}.$$

Then $b \ge 2$ and $m \ge c_0 n/\log n \ge 64$, uniformly over the resource parameters. The saturated cap satisfies

$$b! \le b^b \le n^{1/4} < \lfloor m/32 \rfloor, \qquad \frac{m \log m}{\log b} \le n.$$

Thus it is admissible in Lemma 16. For a fixed constant A covering that lemma, the resulting cost obeys

$$C \le An\left(1 + \frac{2^b}{\kappa}\right) \le An + \frac{A}{K} B \le B,$$

on first choosing $K \ge 2A$ and then $K_0 \ge 2A$ as above. These choices are independent of κ. Rounding the workspace gives $n/m \le n/M_{\mathrm{row}} + 2\log n/\log b$, so $R = O(n/M_{\mathrm{row}} + \log n/\log b)$.

If the budget-dependent term sets b, then $\log b = \Theta(\log(2 + \log(2 + \kappa B/n)))$, including bounded $\kappa B/n$ above the fixed threshold. If the saturated term sets it, $\log b = \Theta(\log\log n)$ and the second call term is $O(g_n)$. This gives the stated maximum. Only the saturated integer cap, workspace, and floor descriptors enter the compiled layout. Arbitrarily large work budgets therefore require no additional precision or resident program bits. The data alphabet remains the fixed one of Lemma 16. □

E Direct proof of the finite-payload frontier

This appendix proves Theorem 8 using only the interfaces, resources, and finite-payload condition from the main text. The local template below converts paid accesses into formulas; the parity bound then turns their size and depth into a work–call constraint. The matching upper construction is in Appendix D. All constants may depend on the fixed architecture, precision, payload, and error caps, but are uniform in $n, B, M_{\mathrm{row}}, \kappa$.

E.1 Proof overview

A generic circuit representation gives the call floor g_n without tracking work. Our formula representation tracks paid accesses, including adaptive routing and stopping, while finite payloads control growth between calls. Together, these connect classical parity formula bounds to the joint resource constraints.

Fix a random tape and prescribe a maximum row access charge for each stage. Lemma 18 turns each local update into a Boolean formula, including source selection, comparisons, tie priority, and masked slots. The inequality $\sum_s \max_i a_{si} \le \kappa C/(a_0 m)$ bounds the total stage charges. Although numerical possibilities can grow within a call, every row, including controllers, returns to its fixed alphabet at the boundary.

A partial evaluation marks a record unknown if it requires an unknown source or exceeds its prescribed charge; boundary projection can erase an invalid record but cannot invent a value. Only output and halt checks enter the accepting formula, so no global test of all rows is needed. Each accepting formula certifies an actual answer, and every execution within the charges has a complete certificate. Combining stopping stages and charge allocations preserves the size bound.

Markov's inequality and averaging over the entire input cube convert expected work and calls into a formula with average error below one half. Applying the parity bound in Lemma 19 to this single formula gives the lower bound; partial padding attains it with every row and head charged.

E.2 Local updates as formulas

Lemma 18 (A local execution template)**.** *Fix a stage, destination, tape, and input-independent metadata. Give each preceding source record, including the destination and at most d designated controllers, an alphabet of at most $K \ge 2$ complete records including $\perp$. Use separate binary labels for these row-specific alphabets, with invalid labels interpreted as $\perp$. There are h declared hard-attention heads. For each known own/controller tuple, the local rule either returns $\perp$ or fixes every head's kernel and complete ordered source list; the total number of physical slots is at most $Q \ge 1$. This determination may not read other input-dependent records. Eligibility and scores use only the query/controller tuple, a candidate source record, and fixed metadata; the source priority is fixed. The final local map uses only that tuple and the selected head outputs, including a selected identity if retained.*

Define a partial update to return $\perp$ if an own/controller record or any listed source is unknown. Otherwise execute the declared update, with the specified default for a head with no eligible source. Then:

1. *At most QK^{d+1} distinct source identities are relevant across all retained own/controller tuples, counting an identity only once.*
2. *The output alphabet, including $\perp$, has size at most*
$$K' \le 1 + K^{d+1}(1+QK)^h.$$
3. *Each output-label bit and its complement have AND/OR formulas with input negations, depth at most four, and occurrence size at most*
$$C_{d,h}(2+Q)^{h+1}K^{d+h+3}.$$
Every copied source-label occurrence is counted, including repeated or masked slots. Known outputs agree exactly with the declared numerical update; no unknown required input becomes a known output.

These are representation bounds, not runtime operations. They are independent of the number of rows, record precision, and imbalance cap. The constants depend only on d, h.

Proof. There are at most K^{d+1} known own/controller tuples u. Each retained tuple determines at most Q slots, so their union contains at most QK^{d+1} source identities. This counts possible gather addresses as well as scan entries. Iterating a descriptor introduces no additional input-dependent choice under the complete-list premise. For a fixed u, each head returns either a fixed default or one of at most QK source-identity/value pairs. Its output and the final local map are deterministic once

these choices are fixed, proving the bound on K'. Distinct choices can produce the same record, which only reduces the count. Take the output label of $\perp$ to be all zero.

We give formulas explicitly. For a source identity j and nonunknown label v, let $E_{j,v}$ be the conjunction asserting that source j has precisely that binary label. It has $O(\log(2K))$ literals. Any specified subset of the known labels is tested by OR-ing at most K such conjunctions, with depth at most two and size $O(K \log(2K))$. This includes the test that a source is known.

Enumerate a known tuple u and, for every head, either a proposed winning physical slot and its source label or the default outcome. For each proposed winner, discard choices whose source is ineligible. Conjoin its label test and, for *every* listed slot, the test that that slot is known and is ineligible or no higher in the score/priority order. The winning score and priority are constants once u and the winner's label and identity are fixed. Hence each comparison is a subset-of-labels test of the preceding form. For a proposed default, require every listed slot to be known and ineligible. A gather uses its certified singleton list. Empty lists use the declared default. Repeated identities may appear in several tests; contradictory label tests reject automatically. Masked slots still receive known-source tests, even though they cannot defeat a winner.

Conjoin these tests with the own/controller label equalities. Each resulting certificate is true exactly when that tuple and those head outcomes describe the partial update. Its conjunction adds one level to the depth-two tests. For each output-label bit, OR the certificates whose numerically evaluated output has that bit equal to one. This adds the fourth level. If any required value is unknown, no certificate accepts and the output is the all-zero label of $\perp$. Fixing both polarities by De Morgan's rules preserves depth and occurrence size.

There are at most $K^{d+1}(1+QK)^h$ certificates. Each uses at most $C_{d,h}(1+Q)K\log(2K)$ occurrences: all comparisons, known-source tests, winner labels, and own/controller labels are included without sharing. Multiplication and $\log(2K) \le CK$ give the stated bound. The fixed numerical comparisons and update merely determine which labels enter these finite tests and which constant output code each certificate receives. The formula never receives an uncomputed numerical value from a truth table or metadata. □

E.3 Parity and formula size

Lemma 19 (Parity correlation of a formula union)**.** *If $F = \bigvee_{h=1}^{H} G_h$ has uniform-average error against parity (possibly complemented) at most $\eta < 1/2$ on N bits, and every G_h has an AND/OR formula with input negations, size at most $S \ge 2$, and depth at most $d+1$, then*

$$N \le C_\eta \log(2H) \left[C\left(1 + \frac{\log(2S)}{d}\right)\right]^d, \qquad d \ge 1.$$

The accepting sets may overlap arbitrarily.

Proof. The depth-uniform criticality theorem of Harsha et al. (2023, Theorem 1.1) gives $\lambda = [C(1+\log(2S)/d)]^d \ge 1$ and

$$\Pr_{\rho\sim\mathcal{R}_p}[\mathrm{DT}(G_h|_\rho) \ge t] \le (p\lambda)^t,$$

where $\mathcal{R}_p$ leaves coordinates free independently with probability p and fixes the rest uniformly. Its size convention is bounded by occurrence count; unary padding handles smaller depths. Put $\gamma = 1/2 - \eta$, $\varepsilon = \gamma/4$, $p_1 = 1/(2\lambda)$, and $t = \lceil \log_2(H/\varepsilon)\rceil$. After $\mathcal{R}_{p_1}$, every G_h has decision-tree depth below t except with probability ε. Their union then has a width-t DNF, with no needed bound on its number of terms. Under a fresh $\mathcal{R}_{p_2}$ restriction, $p_2 = \varepsilon/(5t)$, the switching lemma (Rossman, 2018, Lemma 6) makes this DNF constant except with probability ε.

The unconditioned composition is $\mathcal{R}_{p_1p_2}$. Whenever F is constant and at least one variable survives, its conditional correlation with the target's signed parity χ is zero. Thus $\gamma \le \mathbb{E}[F\chi] \le 2\varepsilon + e^{-Np_1p_2}$. If $Np_1p_2 \ge \log(1/\varepsilon)$ this contradicts $3\varepsilon < \gamma$. Rearranging proves the claim. □

E.4 A direct proof for finite retained payload

Proof of the finite-payload frontier 8. We use only Definition 7, the common execution model, and the local-template and parity-correlation lemmas above. All constants below depend only on the

fixed architecture, precision, prefill, and error caps, and may depend on the fixed β in the final bound. They are independent of $n, B, m, M_{\mathrm{row}}, \kappa$. Assume finite worst-case expected work $\bar{C}$ and calls $\bar{R}$; an infinite call count cannot improve the optimum.

Local input records and fixed boundary alphabets. Apply the simultaneous prefill restriction from Appendix B.1. It leaves $N \geq n/2$ free input bits and makes every initialized mutable or frozen record depend on at most a fixed c of them. Parity on this cube is possibly complemented parity of its free bits. The restriction is fixed before the independent tape. For fixed n, almost-sure termination on every input lets us discard one null set of tapes. Fix a remaining tape temporarily, and let L bound the stages per call.

For row i at boundary r, Definition 7 specifies a fixed alphabet of at most 2^{β} complete records: its metadata are fixed by the program, row, boundary, and tape, and its data range over the prescribed payload alphabet. This includes controllers and retained scratch. Introduce a new symbol $\perp$ for an unknown record. Projection onto this alphabet retains a computed record if it belongs to the alphabet, and returns $\perp$ otherwise; $\perp$ always remains $\perp$. Projection never supplies a guessed numerical value. Frozen records retain their c-local alphabets, of size at most 2^{c}, throughout.

A formula for capped executions. Fix an integer stage cap $J \geq 1$ and a real charge cap $W \geq 0$. For each terminal stage $0 \leq j \leq J$, consider positive integer ceilings $Q_1, \ldots, Q_j$ with $\sum_s Q_s \leq \lfloor W \rfloor$. For $j = 0$ this is the empty vector. Starting from the projected initializers, define partial updates stage by stage. Unknown own or required controller records make a destination unknown. Otherwise they determine its actual declared kernels, row charge, and ordered source lists. Discard a tuple whose row charge exceeds Q_s. For a retained tuple, require every listed source to be known, including masked and repeated scan slots, then perform the declared numerical update. This is the partial update of Lemma 18. At each completed nonterminal call, project every mutable row into its fixed boundary alphabet. A terminal partial call requires no outgoing projection.

These projections are rowwise maps, not a global test that all rows are known. We compile only the designated readout and stopping records and the predecessor records needed to compute them. Conjoin known-record guards for every preceding nonhalt check and for the terminal halt and answer-one check. There are only constantly many designated records per check and constantly many checks per stage. Include the initial halt check; a check absent from the source program contributes no guard. Each resulting certificate asserts a genuine first halt with answer one.

To verify this assertion, induct through a prefix certified not to have halted. Every known initializer equals the source initializer. Known own and controller records therefore choose the source's actual lists. Requiring all listed sources to be known makes the update agree with the source, and projection can only erase its output. Thus each known record equals its source record. The stopping guards make this induction valid up to the first halt. This argument does not require the source to satisfy either cap: missing records outside the checked cones cannot produce a false answer-one certificate.

Conversely, suppose the source first halts by stage J and its actual charges satisfy $\sum_s \max_i a_{si} \leq W$. Choose $Q_s = \max_i a_{si}$ along that execution. At initialization all mutable records belong to their boundary alphabets. Inductively all records remain known through every stage and completed boundary, by the finite-payload promise. The corresponding certificate therefore accepts exactly when the source answer is one. Initial halts use the empty ceiling vector; a completed terminal call may be read before any outgoing projection. In particular, no condition on a fictitious post-halt state is needed.

Formula size. Use binary labels for each row-specific alphabet, reserving zero for $\perp$, and carry both polarities of each label bit. At a block entrance, mutable and frozen alphabets have size at most $K_0 = \max\{2^{\beta} + 1, 2^{c} + 1, 2\}$. Inside a block, form closure alphabets by applying the local-template lemma to all retained own/controller tuples and possible head outcomes. For constants A, λ depending only on the local architecture, their sizes satisfy

$$K_s \leq A(2 + Q_s)^{A} K_{s-1}^{\lambda}.$$

The closure includes complete numerical outputs and any retained winner identity. Arbitrary admitted within-call arithmetic changes which output is assigned to a tuple, but not this bound on the number of outputs. A boundary projection is a fixed truth table on the preceding closure labels, with no additional record arguments.

There are at most L stages between successive projections. Hence a block with total ceiling charge v_b has a formula template for each output-label bit, its complement, or a local stopping guard, of depth $O(L)$ and occurrence size bounded by

$$2 \le T_b \le 2^{A'(\beta+c+1)}(2+v_b)^{A'}$$

for a fixed A'. This bound includes the boundary projection when present. It also holds for a terminal partial block. All copied source occurrences, including those from masked slots, are counted.

Each initializer-label bit has a constant-size formula because it depends on at most c free bits. Let V_b bound the largest unfolded single-label-bit formula through block b, including frozen formulas, and let G_b be the size of the conjunction of the preceding stopping guards. With a fixed constant A'', unfolding gives

$$V_b \le T_b(1+V_{b-1}), \qquad G_b \le G_{b-1} + A''T_b(1+V_{b-1}).$$

The initial bounds are constant, and a geometric-product bound yields logarithmic certificate size $O(1+\sum_b \log T_b)$ and depth $O(j+1)$. There is no factor m from checking unrelated rows: all-row knownness was used for completeness, not as an accepting guard. There are no guessed boundary values.

Put $J_0 = J+1$. Concavity of $\log(2+v_b)$ and $\sum_b v_b \le W$ bound each certificate's logarithmic size by $O(J_0[\beta+c+1+\log(2+W/J_0)])$. The number of terminal-stage and ceiling-vector choices is at most

$$(J+1)\binom{\lfloor W \rfloor + J}{J},$$

whose logarithm is $O(J_0 \log(2+W/J_0))$. OR all these certificates into one AND/OR formula F, with negations only on free input bits. Its accepting sets need not be disjoint. With occurrence size $S \ge 2$,

$$\log(2S) = O(J_0[\beta+c+1+\log(2+W/J_0)]), \qquad \mathrm{depth}(F) = O(J_0).$$

The formula is source-one sound on the whole restricted input cube and agrees with the source on executions satisfying both caps. The OR changes the size bound by the displayed ceiling count only; no runtime schedule, cap test, or state is supplied to the source.

Expected budgets and a single parity formula. Set $\varepsilon = (1/2-\delta)/4$ and fix, before choosing the tape,

$$r = \max\{1, \lceil \bar{R}/\varepsilon \rceil\}, \qquad J = Lr, \qquad W = \frac{\kappa \bar{C}}{a_0 \varepsilon m}.$$

Every source execution obeys $\sum_s \max_i a_{si} \le \kappa C/(a_0 m)$ by the stagewise imbalance condition and the charged all-row work. Markov's inequality gives, for every input, probability at most ε of exceeding r calls and at most ε of exceeding W total maximum-row charge. Whenever neither event occurs, the formula just constructed equals the source answer. Its error indicator is therefore at most the sum of the source-error indicator and these two cap-failure indicators. Average first over all uniform completions of the prefill cube and then over the independent tape. Some fixed tape gives F uniform-average parity error at most $\eta = \delta + 2\varepsilon < 1/2$. We have not conditioned on inputs with cheap executions, and do not assert per-input caps for this tape.

Apply Lemma 19 to this single formula, with $H = 1$. Pad its depth to $d+1$, where d is a sufficiently large fixed multiple of J_0; the additional $O(J_0)$ unary gates preserve the size bound. The size and depth bounds imply

$$\log N = O\big(J_0 \log\big(2+\beta+c+\log(2+W/J_0)\big)\big).$$

The independent finite-record floor (Theorem 14) gives $\bar{R} = \Omega(n/m + g_n)$, so $J_0 = \Theta(\bar{R})$ for sufficiently large n. Absorbing the fixed c and the cap constants yields

$$\log n = O\left(\bar{R} \log\left(2+\beta+\log\left(2+\frac{\kappa \bar{C}}{m\bar{R}}\right)\right)\right).$$

In particular $m\bar{R} = \Omega(n)$. For $\bar{C} \le B$ and fixed β, the last bound gives

$$\bar{R} = \Omega_\beta\left(\frac{\log n}{\log(2+\log(2+\kappa B/n))}\right).$$

Combining it with the capacity and g_n floors proves the required lower bound for every $m \le M_{\mathrm{row}}$, hence for the optimum.

Matching construction. Corollary 17 supplies the upper bound with the required work, imbalance, architecture, precision, and storage budgets. Its retained data have a fixed constant alphabet: data heads return bits, numerical scratch is cleared, and roles, addresses, cursors, and controller clocks are input-independent at each fixed boundary. Initialization has the same property. Thus every complete row, including its controller fields, satisfies Definition 7 for the fixed threshold $\beta \geq \beta_*$. This is an upper bound inside the finite-payload class itself. Together with the direct lower bound it proves the frontier. □

The displayed work and imbalance regimes. Write $\ell_\kappa = \log(2+\kappa B/n)$. For fixed κ, substituting the three work budgets gives, respectively, $\ell_\kappa = \Theta(1)$, $\ell_\kappa = \Theta(\log\log n)$, and $\ell_\kappa = \Theta((\log n)^\alpha)$. At linear work, the same orders follow by taking respectively constant κ, $\kappa = \lceil (\log n)^a \rceil$, and $\kappa = \lceil \exp((\log n)^\alpha) \rceil$, where $a > 0$ and $0 < \alpha < 1$ are fixed. With $M_{\mathrm{row}} = n$, the corresponding call orders are $\Theta(\log n)$, $\Theta(\log n/\log\log\log n)$, and $\Theta(g_n)$. These substitutions compare prescribed classes; they do not change κ while optimizing a single class over B.

E.5 An open case for unrestricted numerical state

The finite-payload assumption leaves a concrete question. Retain the common model, per-input error bound, and $O(n\log n)$ stored/program bits, but allow full admitted numerical records. At row capacity $M_{\mathrm{row}} = n$, balanced execution $\kappa = 1$, and work budget $B = Kn\log n$ for sufficiently large fixed K, let $R^*_{\mathrm{full}}(n)$ be the fewest worst-case expected calls. Theorem 14 and the finite-payload construction of Corollary 17 give, respectively,

$$\Omega\left(\frac{\log n}{\log\log n}\right) \leq R^*_{\mathrm{full}}(n) \leq O\left(\frac{\log n}{\log\log\log n}\right).$$

The upper bound remains valid because a finite-payload implementation also belongs to the unrestricted class. Whether numerical retained state can improve it at this budget is open. The optimization permits choosing fewer than n evaluated rows; fixing all n rows would instead force $\Omega(\log n)$ calls by Theorem 5.

F The boundary of the row-cost balance assumption

F.1 When row costs may differ arbitrarily

At $\kappa = M_{\mathrm{row}}$ the balance constraint is vacuous: expensive scans may be concentrated without a balance restriction. The resulting optimum also permits full numerical state:

Proposition 20 (Unequal row costs)**.** *Retain the setup and its stored/program budgets, but remove the row-cost imbalance inequality. For sufficiently large fixed K, every $B \geq Kn$, and $1 \leq M_{\mathrm{row}} \leq n$, the optimal worst-case expected parity call count is*

$$R^*_{\mathrm{unequal}}(n, B; M_{\mathrm{row}}) = \Theta\left(\frac{n}{M_{\mathrm{row}}} + \frac{\log n}{\log\log n}\right).$$

The lower bound allows full numerical records, without a retained-state promise. Exact loops attain the bound with fixed source banks and constant retained data, while every row and declared head executes.

Linear work now reaches the call floor, subject to row capacity. In the attaining construction, a few rows use large banks while the others execute small ones, so shared parameters and all-row execution do not enforce balance. The proof follows below; Section 8 distinguishes call count from elapsed time. We prove Proposition 20 by retaining the factorial reduction of Appendix D and changing its physical scan banks. This isolates the role of balanced costs: the numerical operations, local access rules, and all-row execution remain as specified in Section 3.

Lemma 21 (Fixed banks with unequal costs)**.** *For $1 \leq m \leq n$, an exact constant-width loop with exactly m evaluated rows computes parity with*

$$R = O(n/m + g_n), \qquad C = O(n + mg_n), \qquad g_n = \log n/\log\log n.$$

Its block depth and head count are constant, its precision is $O(\log n)$, and its stored and program bits are $O(n \log n)$. Banks and kernel types are fixed by destination and stage. Every declared head executes, including at padding rows and unused roles. Ordinary records have a fixed retained-data alphabet at call boundaries.

Proof. For $m \geq 64$, put $N_0 = \lfloor m/32 \rfloor$ and let r be least with $(r+1)! \geq N_0$. Specialize the construction of Appendix D to $b = r + 1$. Its reduction has successive arities $2, 3, \ldots, r + 1$ and no capped tail. Level j contains $N_j = \lceil N_0/(j+1)! \rceil$ parity modules, each with 2^j literal helpers and one aggregator. The base accumulators incorporate disjoint strided input groups using the same clamped positional gathers and guarded XOR updates.

Keep each helper's physical bank equal to its actual child group plus the two constant sentinels. Keep an aggregator's bank equal to its literal helpers. The two literal scans include every listed source, including masked slots. At unused roles, use a fixed eligible zero singleton and discard its result. All input and preservation heads still execute their certified gathers. Thus the only change from the earlier construction is to omit padding of every scan bank to common cardinalities. No rows, heads, or calls are skipped.

The shared block refreshes the bases, then all literal helpers, then all aggregators. Helpers read newly refreshed bases and preceding-call aggregators, exactly as before. The same induction shows that after $B_0 = \lceil n/N_0 \rceil$ loading calls and at most r propagation calls, the fixed root row holds parity. Store a clock and the fixed horizon $B_0 + r$ in protected fields of that designated row; incrementing and testing them adds constant work and width. Stopping therefore uses only a designated record, and the affine binary-score readout reads the root's Boolean value. No separate scheduler or mutable row is needed for the clock.

The factorial row bound gives fewer than $9N_0 < m$ computational rows. Pad to exactly m rows, whose fixed singleton heads and maps also execute. The total literal and aggregator scan incidence in a call is at most

$$\sum_{j=1}^{r} N_j 2^j (2j + 7) = O(N_0).$$

Replacing N_j by $N_0/(j+1)!$ gives a convergent factorial series of order $O(N_0)$. The ceiling errors contribute $O(r2^r) = O(N_0)$, using $r2^r \leq 4r! < 4N_0$. Unused-role singleton scans, all-row gathers, protected-field updates, and scratch resets add $O(m)$ work per call. The data heads project Boolean values rather than winner identities; scratch is cleared at the boundary. Roles, cursors, and clocks are input independent at each fixed boundary. These observations establish the claimed constant retained-data alphabet.

Fixed adjacency lists and eligibility masks have the same $O(m)$ incidence bound. Enumerating their entries, validating source ranges and masks, assigning role tags, and building the positional-gather descriptors take $O(m)$ word work and words. Pattern enumeration is charged by its emitted mask entries; no factorial or pattern-extraction primitive is needed during a call. Raw-input records and squared positions require $O(n)$ additional preparation work and storage. All pointers, cursor values, score intermediates, and the stopping horizon have polynomial magnitude in n, so one fixed precision coefficient suffices. Resident parameters, double buffers, live-list iterators, and fixed metadata together use $O(n \log n)$ bits.

Since $r = O(g_n)$, total work is $O(n + m(B_0 + r)) = O(n + mg_n)$ and calls are $O(n/m + g_n)$. For $m < 64$, use one running-parity accumulator and evaluate the remaining singleton roles on every call. Its n calls and $O(nm) = O(n)$ work meet the same bounds uniformly over these bounded workspaces. □

Proof of Proposition 20. Theorem 14 gives $\bar{R} = \Omega(n/m + g_n)$ without a balance or retained-state assumption. Optimizing over $m \leq M_{\text{row}}$ preserves the lower bound $\Omega(n/M_{\text{row}} + g_n)$, independently of the work budget. For the upper bound, choose an integer $m = \Theta(\min\{M_{\text{row}}, n/g_n\})$ in Lemma 21. It uses $O(n)$ total work and $O(n/M_{\text{row}} + g_n)$ calls. A sufficiently large fixed K makes the construction feasible at every $B \geq Kn$.

For completeness, fixed-workspace optimal work is $\Theta(n + mg_n)$ under this evaluator. The raw-input argument gives $\bar{C} = \Omega(n)$, and every call updates all m rows, so $\bar{C} = \Omega(m\bar{R}) = \Omega(mg_n)$. The

construction matches both terms. These are work and semantic-call bounds; a scan still incurs its primitive execution time. □

G PARALLEL EXECUTION TIME

Classical parallel evaluation separates operation counts from time with limited processors (Brent, 1974); shared-memory machines also exhibit depth–width trade-offs (Vishkin & Wigderson, 1985). The consequence below distinguishes primitive execution time from semantic calls under our stated CREW rules. Retain the fixed architecture, precision, input, and accuracy conventions, but allow arbitrary row-cost imbalance. Execute the declared operations on a synchronous shared-memory machine with $1 \le \Pi \le n$ processors, concurrent reads, and exclusive writes. Each processor executes at most one finite-word primitive or word access per time unit; each primitive accesses constantly many words. CoT respects online token dependencies, and loops respect synchronous layer dependencies. Initially each raw bit occupies its own cell and all other state is input independent. Encoding, prefill, layout construction, and readout are timed and charged.

Corollary 22 (Time at a work budget)**.** *For sufficiently large fixed K, every $B \ge Kn$, and $1 \le \Pi \le n$, the optimal worst-case expected parity times are*

$$\tau^*_{\mathsf{CoT}}(n, B, \Pi) = \Theta(n), \qquad \tau^*_{\mathsf{Loop}}(n, B, \Pi) = \Theta(n/\Pi + \log n).$$

The loop chooses a fixed $1 \le m \le n$ workspace. Stored and program bits are $O(n \log n)$, with all caps uniform in B, Π. Full admitted numerical records are allowed; the finite-payload condition of Theorem 8 is not required.

Proof. Work is at most Π times elapsed time. The raw-input work lower bound therefore gives $\Omega(n/\Pi)$ time. Online CoT also needs $\Omega(n)$ sequential token calls by Theorem 4.

For loops, the classical probabilistic CREW parity theorem gives $\Omega(\log n)$ time at any fixed horizon with per-input success greater than $1/2$ (Dietzfelbinger et al., 1994). Kutyłowski (2005, Theorems 2.3 and 4.1) states the result with its degree proof. That machine permits arbitrary local computation and one shared read/write per processor step. Our primitives are simulated with constant overhead by separating their constantly many accesses. All preprocessing is included, so the initial raw-bit layout matches. For worst-case expected time $\bar{\tau}$, truncate at $\lceil \bar{\tau}/\eta \rceil$ for fixed $0 < \eta < 1/2 - \delta$. Markov bounds timeout error by η on every input; recording completion and returning a fixed answer on timeout gives a hard-horizon computation with error below $1/2$. Thus $\bar{\tau} = \Omega(\log n)$. The two worst-case lower bounds need not have the same maximizing input.

For the upper bound, let $\ell = \lceil \log_2 n \rceil$ and choose the greatest power of two $m \le \Pi$ satisfying $m\ell \le n$, handling bounded n directly. The binary loop of Appendix C uses $O(n/\Pi + \log n)$ calls and $O(n)$ work. Assign one processor per mutable row. Its certified gathers and constant-width maps take constant time per stage; double buffers give exclusive writes. Integer doubling and comparisons construct the parameters in $O(\log n)$ serial time. Parallel encoding, squared positions, and initialization take $O(n)$ work and $O(n/\Pi + \log n)$ time, including any needed broadcasts. Input, states, processor registers, and program fit $O(n \log n)$ bits. The running-parity CoT has the same prefill order and n constant-time token steps. Both constructions use at most Kn work and remain feasible for every larger B, whereas the time lower bounds do not depend on B. □

Unit-cost word access and arithmetic are material assumptions. The result does not price a scan as a constant-time operation, predict hardware latency, or cover arbitrary task-specific recompilation of online CoT into another algorithm. It uses a classical parallel lower bound rather than a new machine-complexity argument.

H AN ORDERED-GROUP TASK AT THE SAME WORK ORDER

Finite-group composition is an established task for studying looped Transformers: Saunshi et al. (2025, Theorem 5.1) give a logarithmic-loop construction. The result below applies our resource comparison to the *single Boolean answer* indicating whether an ordered product is the identity. It neither requires an output trace of prefix products nor claims a new group-expressivity result. Its

lower bound transfers from parity; the construction makes order, workspace, and executed work explicit.

Task and binary input interface. Fix a finite group G with identity e and an element $a \neq e$ satisfying $a^2 = e$. Our concrete choice is $G = S_5$ with $a = (1\ 2)$. Put $q_G = |G|$ and $d_G = \lceil \log_2 q_G \rceil$. Fix distinct d_G-bit codes for the group elements, assigning the all-zero code to e and the code with only its last bit set to a. Unused codes decode to e. This convention defines a total function on $N = d_G n$ separately stored raw bits:

$$f_{G,n}(x) = \mathbf{1}[g_1 g_2 \cdots g_n = e], \qquad g_j = \mathrm{dec}_G(x_{d_G(j-1)+1}, \ldots, x_{d_G j}).$$

Products are evaluated in the displayed order. The group, coding table, and all architecture constants are fixed before n and the budgets vary. Since d_G is constant, $N = \Theta(n)$ and $O(N \log N) = O(n \log n)$; no group symbol is treated as an uncharged oracle for its bits.

Proposition 23 (Same-work comparison for ordered group composition)**.** *Fix G as above, per-input error $\delta < 1/2$, and a constant row-cost imbalance cap κ_0. Under the deterministic prefill, hard-attention, whole-record precision, and other task-comparison assumptions of Section 3, with fixed architecture caps sufficient for the constructions below, the following hold for $n \geq 2$ and $1 \leq m \leq n$.*

1. *The optimal CoT token count is $\Theta(n)$, including sampled decoding and adaptive stopping. An exact decoder attains $\Theta(n)$ tokens and $\Theta(n)$ total work.*
2. *The optimal loop work at fixed physical workspace m is $\Theta(n + m \log n)$. At this work order, the fewest calls are $\Theta(n/m + \log n)$. For every m, an exact shared-block construction uses $O(n/m + \log m)$ calls and $O(n + m \log n)$ work.*
3. *For sufficiently large fixed K and capacity $1 \leq M_{\mathrm{row}} \leq n$, the optimal loop call count under total work Kn is $\Theta(n/M_{\mathrm{row}} + \log n)$.*

The exact upper bounds use $O(n \log n)$ stored and program bits. Their loops belong to $\mathcal{F}_\beta$ from Definition 7 with $\beta = \log_2 q_G$. The lower bounds allow full admitted numerical records and do not assume finite retained payloads. All constants may depend on G, the fixed architecture and precision caps, κ_0, and δ, but not on n, m, M_{row} or the varying work budget.

H.1 Finite-payload preservation

The following closure criterion verifies membership in $\mathcal{F}_\beta$ through local writing rules. It will certify the ordered-group construction below.

Operational criterion. Fix a finite label alphabet Λ and a constant number s of data registers per row, before the input and work budget. The following rules cover every retained field, including controller fields.

1. Data registers are initialized with labels from Λ, using a charged fixed local input decoder or constant labels. Each subsequent write copies a data register, writes a constant label, or applies a fixed finite table $\Lambda^u \to \Lambda$ of constant arity. The table may be selected by the metadata.
2. An attention value passed to a data register consists of a fixed number of such labels. Its winner may depend on input-dependent scores or eligibility. A winner address or other numerical side output is not passed to a retained data or metadata register.
3. Metadata are initialized independently of the input and updated only from metadata and fixed program constants. They determine the gather addresses, complete declared bank lists, and update phases.
4. Auxiliary numerical calculations use a fixed number of scratch fields in these same rows. Scratch fields and discarded data buffers are initialized to prescribed constants and reset to those constants by the end of every call. No other input-dependent fields survive a call boundary.

All attention, table operations, resets, and metadata updates are executed and charged under the common model. The rules hold for a total continuation after any stopping time.

Lemma 24 (Closure of finite payloads)**.** *A program satisfying the finite-payload criterion in Section H.1 has input-independent metadata and at most $|\Lambda|^s$ possible complete records per row at every initialization and completed-call boundary, once the program, row, boundary, and independent tape are fixed. Consequently it belongs to $\mathcal{F}_\beta$ from Definition 7 with $\beta = s \log_2 |\Lambda|$.*

Proof. Fix the independent tape and induct over executed stages. Initialization puts every data register in Λ and every auxiliary field at its prescribed constant. Metadata remain input independent because no permitted metadata write reads a data value. An attention winner selects already typed payloads; changing its identity does not enlarge the payload alphabet. Every other data write preserves Λ by the declared finite table. At each completed boundary, scratch and discarded buffers again have their prescribed values. Thus only the s labels can vary in a complete row record. Their prescribed alphabet Λ^s has size $|\Lambda|^s = 2^\beta$, including any retained controller data. Together with the input-independent metadata, this is exactly Definition 7. The same invariant holds for the declared continuation after stopping. □

This criterion permits content-dependent attention selection. It does not cover retaining arbitrary winner identities or unconstrained numerical accumulators. The ordered-group implementation below uses certified gathers and one group-valued register, so $s = 1$ and $\Lambda = G$ suffice; its metadata carry positions and clocks, rather than additional unbudgeted mutable rows.

H.2 Exact primitives and charged input decoding

Represent $g \in G$ by its one-hot vector $\mathbf{e}_g$. A fixed-size ReLU network computes multiplication exactly: for one-hot inputs u, v,

$$\mathrm{Mul}(u, v)_c = \sum_{ab=c} \mathrm{ReLU}(u_a + v_b - 1), \qquad c \in G. \tag{8}$$

Exactly one pair contributes. The q_G^2 hidden units and their weights are constants independent of n and the budgets. For a raw code $z \in \{0,1\}^{d_G}$ and fixed code w, its indicator is

$$I_w(z) = \mathrm{ReLU}\left(1 - \sum_{j:w_j=0} z_j - \sum_{j:w_j=1} (1 - z_j)\right).$$

Summing indicators over the codes assigned to one element gives its one-hot decoder, including the unused codes assigned to e. Thus decoding is also a fixed-size ReLU map.

At the raw position ending a code block, d_G certified positional gathers read that block's bits; they are causally available. Other raw positions execute the same heads with certified clamped addresses and discard their unused outputs. The endpoint activations supply the decoded frozen group records and, for CoT, the corresponding input cache values. This is a fixed-depth deterministic prefill on the original N positions. A frozen record uses the activation at one fixed endpoint, as allowed in Section 3; no additional encoder position or global input summary is introduced. Constructing the raw positions, squared-address fields, endpoint records, and all later padding costs $O(n)$ word work and $O(n \log n)$ bits. The constant factor includes the decoding table and every prefill head.

H.3 A compact loop preserving the product order

For an arbitrary physical workspace m, let k be its greatest power-of-two predecessor. Use the first k rows as logical rows, indexed $i = 0, \ldots, k-1$, and put $b = \lceil n/k \rceil$ and $n' = kb < n + k$. The decoded frozen interval is padded with $g_j = e$ for $n < j \le n'$. Its padding is input independent and is added after the charged encoder. Every logical row initially has accumulator $a_i = e$, cursor $\ell_i = ib + 1$, common clock $t = 0$, and common reduction offset $h = 1$. Metadata include k, b and the fixed stopping boundary $r_* = b + \log_2 k$.

Each call executes two certified gathers per destination: one returns $v_i = g_{\min\{\ell_i, n'\}}$, and the other returns the preceding accumulator at $j_i = \min\{i + h, k - 1\}$. Define w_i to be this second value when $i + h < k$ and e otherwise. The head executes even in the latter case; an input-independent local gate replaces its unused value. With $\lambda = \mathbf{1}[t < b]$ and $\mu = \mathbf{1}[b \le t < r_*]$, update

$$a_i' = \begin{cases} a_i v_i, & \lambda = 1, \\ a_i w_i, & \mu = 1, \\ a_i, & t = r_*. \end{cases} \qquad \ell_i' = \ell_i + \lambda, \quad h' = h + \mu h, \quad t' = \min\{t + 1, r_*\}. \tag{9}$$

Here h only doubles in a reduction call, reaching k at the last one; for $k = 1$ there are no reduction calls and h stays one. Both heads still execute in every loading, reduction, and continuation call.

Every extra physical row duplicates logical row zero's metadata and updates. The mutable source interval is always the first k rows, so these extra destinations never change its contents.

Integer comparisons, clamping, and bounded conditional gates use the fixed ReLU primitives from Appendix C, while (8) supplies both candidate products. Each candidate product is evaluated on every row, including when its phase output is unused. All addresses are obtained from the destination's own metadata. Quadratic positional scores certify the two gathers; there is no data-dependent source-list lookup. A fixed sequence of preparatory, attention, table, gating, and reset sublayers forms one shared block. Synchronous old/new values occupy distinct constant-width fields of the same m rows. After the new accumulator is written, the discarded fields and every input-dependent temporary are explicitly cleared. These resets cost $O(m)$ work per call and require no extra mutable rows.

After loading, row i contains the contiguous chunk product

$$A_i = g_{ib+1} g_{ib+2} \cdots g_{(i+1)b}.$$

After r reduction calls, induction gives the ordered interval invariant

$$a_i^{(r)} = A_i A_{i+1} \cdots A_{\min\{i+2^r-1,k-1\}}. \tag{10}$$

At the next call, the interval beginning at $i + 2^r$, if present, is adjacent to and disjoint from the first interval. The update appends it on the right; otherwise it appends e. Associativity proves the invariant without commuting any elements. At $r = \log_2 k$, row zero holds $g_1 \cdots g_n$, and its identity coordinate gives the Boolean answer through the fixed affine readout. In later calls the saturated clock holds the answer fixed while all heads and resets continue.

Every stage has the same fixed number of gathers on every destination; all other sublayer widths are fixed. Thus each call costs $\Theta(m)$ word work and its row-cost cap is one. In particular, there are no unpaid scan banks or omitted inactive rows. Because $k > m/2$,

$$R = b + \log_2 k = O(n/m + \log m), \qquad C = O(n + mR) = O(n + m \log n).$$

All cursors and clocks have magnitude $O(n)$, all squared-address scores have polynomial magnitude, and the multiplication intermediates have constant magnitude. A fixed $O(\log n)$ whole-record coefficient therefore suffices. Frozen records, raw input, buffers, constant tables, and the $O(m)$ metadata records and descriptors occupy $O(n \log n)$ stored and program bits, with $O(n)$ preparation work. Only one group label varies per retained row. The reset rule and Lemma 24 give membership in $\mathcal{F}_{\log_2 q_G}$ at every boundary, including continuation after stopping.

H.4 A CAUSAL CoT IMPLEMENTATION AND THE LOWER-BOUND TRANSFER

A CoT may choose the ordinary running-product algorithm. Start from a delimiter carrying e. At step j, a positional gather reads the decoded group at raw endpoint $d_G j$ in the prefixed input cache. The current token supplies the preceding product. The fixed table (8) emits the next product token. After n such steps, one final token reports whether that product is e; all unused heads execute with clamped positional gathers. Position-type and final-step flags are input independent. A fixed-depth decoder uses one sublayer for the input decoder and a later sublayer for this group gather and table, with the ordinary layerwise KV cache. It has no separate recurrent accumulator. The $n + 1$ generated tokens, causal prefill, and readout cost $O(n)$ total word work and $O(n \log n)$ bits. Although this particular algorithm emits intermediate products, the task and its lower bound require only the final Boolean answer.

Lower bounds and completion of Proposition 23. In each raw code block, fix the first $d_G - 1$ bits to zero and leave only the last bit x_j free. These valid encodings represent $g_j = a^{x_j}$, and hence

$$f_{G,n}(x) = 1 - \bigoplus_{j=1}^{n} x_j.$$

This is a literal input restriction, not an additional encoder or a duplication of input variables. It preserves the given architecture, deterministic prefill, cache interface, stopping rule, error guarantee, mutable row count, and executed costs.

To handle arbitrary admitted contextual prefill on these restricted inputs, use the prior-restriction version of the depth-reduction lemma (Amiri et al., 2025, Lemma B.3). Initially the free coordinates

are a fixed fraction $1/d_G$ of the $N = d_G n$ raw positions. Iterate the lemma through the fixed encoder depth, reducing this fraction only to $1/(2d_G)$. One obtains a single restriction leaving at least $n/2$ free coordinates and making every original frozen/initial record a function of a constant number of them. The constants can depend on G and the fixed encoder caps. Complementing the answer changes none of the unread-bit or query-simulation arguments.

The proofs of Theorem 4 and Theorem 5 now apply on this subcube, with n replaced by a constant multiple wherever raw input length occurs. They yield $\bar{T} = \Omega(n)$, $\bar{C} = \Omega(n)$, and, at fixed constant imbalance,

$$\bar{C} \geq c(n + m \log n), \qquad \bar{R} \log \left(1 + \frac{\bar{C}}{m\bar{R}}\right) \geq c \log n.$$

The contextual update-volume argument also gives the n/m call floor when it is nonconstant. As in the proof of Corollary 6, these inequalities force $\Omega(n/m + \log n)$ calls at optimal-order work. At linear work they force $m = O(n/\log n)$ and $\Omega(n/M_{\text{row}} + \log n)$ calls.

The preceding exact constructions attain the work and fixed-workspace call bounds. For the capacity statement, choose a power of two within a constant factor of $\min\{M_{\text{row}}, n/\log n\}$, using one row if necessary. Then $m \log n = O(n)$ and the loop uses $O(n/M_{\text{row}} + \log n)$ calls. Choosing K to cover the fixed decoding, table, and execution constants completes the upper bound. □

Scope of the application. The ordered product requires preserving the input order even though only one bit is returned. This provides a task-level bridge to prior loop theory under the same work accounting. The lower-bound reduction uses the involution, not non-solvability of S_5 or a separation of classical complexity classes. This proposition proves the same-work comparison only; it does not assert a full group work-budget frontier, a dense-attention FLOP bound, or a guarantee about trained-model accuracy.

I Assumptions across results

Task result	Retained numerical state	Row costs
Linear-work comparison	Full admitted records	Constant cap κ_0
Main frontier	Fixed finite payload	Variable cap κ
Execution boundary	Full admitted records	Arbitrary imbalance
Parallel-time comparison	Full admitted records	Arbitrary imbalance

Table 4: Additional conditions by result. All cases retain hard attention, fixed architecture/precision caps, charged preparation, local access rules, and all-row execution. Simulations separately expand resources.

J Finite execution study: methods and complete comparisons

This appendix gives the implementation and measurement details behind Section 7. The study supports a theoretical contribution within a limited single-device budget. It evaluates explicit programs from our constructions and measures how their dependency structure, workspace, and scan padding translate into execution costs; it does not train Transformer models or compare learned reasoning accuracy. The budget is devoted to correctness checks and repeated measurements of two construction mechanisms: workspace allocation and scan padding. The matching resource bounds are established by the proofs, independently of these measurements.

J.1 Programs, inputs, and correctness checks

Workspace comparison. Experiment A uses lengths $n \in \{64, 128, 256, 512, 1024, 2048, 4096\}$ and batches of 1 and 32. At every length it compares the cached running-parity CoT program, a compact binary loop, and a full binary loop from Appendix C: 42 configurations in total. CoT retains the input, delimiter, and generated records in a cache of length $2n + 1$; each step reads the previous generated value and the next input value. It has no separate parity accumulator. The

compact loop uses the largest power of two $m \le n/\lceil \log_2 n \rceil$; the full loop uses $m = n$. Both binary heads and both phase candidates execute on every row on every call, including the unused candidate. Their respective horizons are n tokens and $R = n/m + \log_2 m$ calls. The compact horizons are $11, 12, 13, 21, 22, 23, 24$ across the seven lengths; full-loop horizons are $7, 8, 9, 10, 11, 12, 13$.

Padding comparison. Experiment B fixes $n = m = 4096$ and $N_0 = 128$ base accumulators. It uses the capped reduction graph of Theorem 15, with arity cap $b \in \{2, 4, 6\}$ and the partial padding of Lemma 16 at requested imbalance caps $\kappa \in \{1, 4, 32\}$. With both batch sizes, this gives 18 configurations. All have 32 loading calls; the graph adds respectively 7, 5, or 5 reduction calls. Every call evaluates all 4096 rows in the base, helper, and aggregator stages. Helpers read newly loaded bases and previous-call aggregator values; aggregators then read the new helper values. Changing padding preserves the logical graph, arithmetic maps, and horizon. Physical scan lengths are at least their live lengths and the prescribed padding floors. Actual imbalance is computed from the resulting stage charges, rather than identified with the cap.

Experiment A instantiates the upper constructions for Theorems 4–5 and Corollary 6. Experiment B isolates the padding mechanism used in Theorem 8 and Proposition 20. The finite sizes do not test the asymptotic lower bounds or establish an empirical optimum over all admitted programs. In particular, 32 loading calls dominate the tested capped horizons, so the study does not fit the nested-logarithm frontier.

Numerics and validation. The CUDA evaluator implements payload maps with FP32 ReLU operations, without mixed precision or fast math. Metadata and certified clamped addresses use checked signed 64-bit integers; positional keys are physically stored. Data retained at call boundaries are Boolean, and control trajectories are input independent. Each request resets the cache, both state buffers, control fields, and scratch. Three input suites use seeds 2027, 314159, and 271828, with 256 random examples per length per suite plus structured inputs. Configuration order uses seed 5090. Targets come from an independent CPU XOR oracle that is unavailable to the evaluated programs.

The final-implementation GPU gate passed 106,494 exhaustive construction/input pairs, 69,120 primary-suite checks, and 2,930,480 intermediate-value comparisons with zero observed mismatches. The small binary tests exhaust lengths through 12 at every physical workspace. Trace checks cover protected records, source versions, Boolean payloads, input-independent controls, and the full capped pipeline. Poisoned resets and opposite-parity replacement inputs check for stale state and captured answers. Cross-padding traces agree on the logical states. Separate CPU checks exercise the mutual-simulation constructions (Theorem 3, Lemmas 9 and 10), including 128 additional seeded source cases and nonzero initialization/readout circuits. These finite checks support implementation correctness; they do not replace the simulation proofs.

J.2 What is counted and timed

Three accounting levels. A source visit counts one physical source slot of an attention operation, including masked and dummy slots. For one request, the counts are $2n$ for CoT and $n + 2mR$ for the binary loops at the tested powers of two; these include prefill/encoding visits. Capped counts are $n + R(3m + \sum \text{scan lengths})$, with the sum taken over physical scan banks. Formal row charges also contain their baseline term $1 + \sum_h k_h$. The implementation ledger separately counts local maps, scalar accesses, temporary storage, and launches. Neither visits nor these grouped charges are FLOPs or a certified complete finite-word work count: the full lowering of preparation and local scaffolding is not certified. Consequently, we make no exact all-in equal-budget claim for the measured implementations.

The primary backend stores live source lists and compact dummy-interval descriptors and executes custom CUDA kernels. All declared rows, heads, inactive candidates, and padding remain evaluated. Source, PTX, and SASS audits check preservation of their paid load instructions and arithmetic in the exact timed binary. A separate audit observed 18,848 host kernel submissions across the 60 configurations and checked their final outputs. These checks do not measure dynamic instructions or memory traffic. Hardware counter access was denied; cache hits and DRAM bytes remain unobserved. In particular, preserved volatile loads do not imply distinct DRAM transactions.

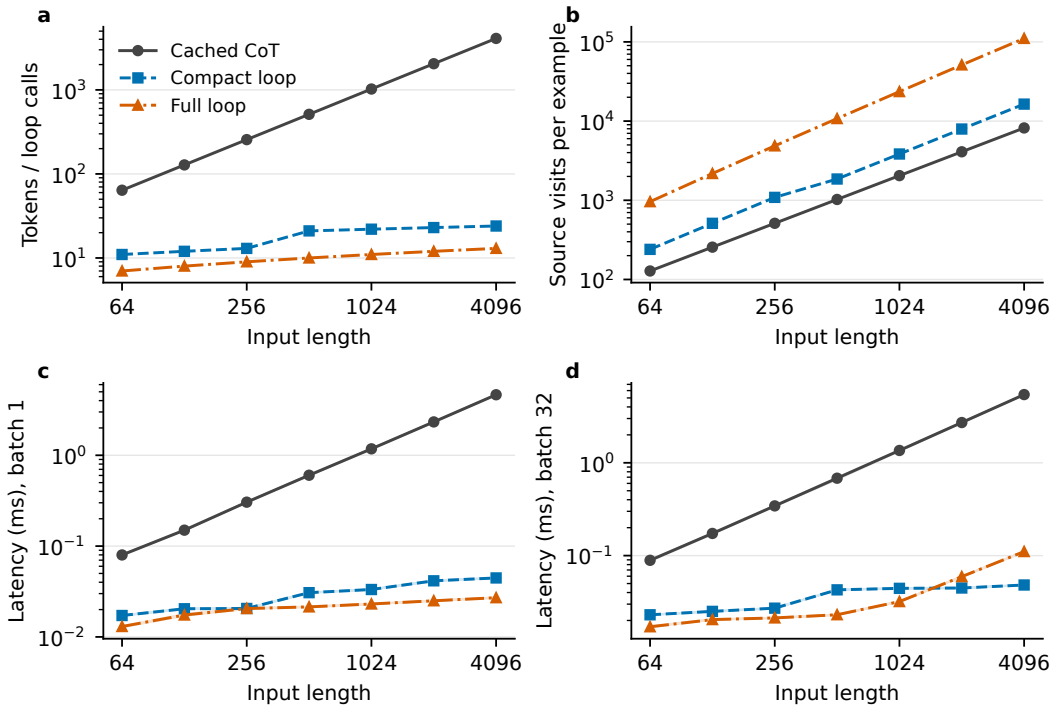


Figure 6: Workspace comparison for the three exact parity programs. Counts are deterministic construction quantities per example, including prefill/encoding visits. Timings are complete-request CUDA Graph measurements on one RTX 5090 D v2, with separate panels for batches 1 and 32; points and uncertainty show the median and interquartile range of 15 block means. The compact loop uses the largest power of two at most $n/\lceil \log_2 n \rceil$ rows; the full loop uses n rows. Fewer calls and fewer visits give different rankings, and the ranking of the two loops by latency changes with batch size.

Hardware and timing protocol. Measurements use one NVIDIA GeForce RTX 5090 D v2 with 25,642,467,328 addressable bytes, driver 595.79, Python 3.12.14, PyTorch 2.10.0+cu128, and NVRTC 12.8 targeting compute capability 12.0. Primary CUDA-event intervals enclose complete, warmed, device-resident requests under a common CUDA Graph replay policy: reset, encoding or prefill, every token/call, and readout are included. Host preparation, allocation/upload, compilation, capture, transfers, fresh-process first requests, and host replay times are recorded separately. They are not added to each warmed request. No cold-cache policy is claimed.

Each input suite has ten complete warmups and five timing blocks. The pilot freezes at least 20 repetitions per block, targeting 0.2 seconds per block within the resource limits. All 60 configurations completed, giving 900 block means. We report their median and first/third quartiles over the 15 block means per configuration; these repeated measurements on one GPU are not independent hardware replications. Batch latency is the whole-batch duration. Dividing it by batch size would give an amortized cost, not single-request latency. Recorded occupied-device wall time is 16.59 minutes across validation, profiling attempts, pilot, core measurements, and supplements; this is reservation time, not measured GPU-active time.

Memory. Logical tensor bytes include input and frozen state, positional keys, token/KV records, both latent buffers, metadata, scratch, descriptors, sentinels, and dummy pools. They are distinct from observed allocator peaks, which are measured in separate per-configuration workers and exclude some driver and graph storage. The loop's complete memory therefore includes the retained input as well as its mutable workspace.

Batch	Program	Median ms	Q1–Q3 ms	Logical MiB	Peak MiB
1	CoT	4.64075	4.62682–4.66652	0.2539	0.2573
1	Compact loop	0.04461	0.04449–0.04476	0.1075	0.1108
1	Full loop	0.02702	0.02699–0.02719	0.4883	0.4917
32	CoT	5.43408	5.39861–5.44110	8.1261	8.1284
32	Compact loop	0.04823	0.04806–0.04827	3.4376	3.4409
32	Full loop	0.11051	0.11049–0.11052	15.6251	15.6284

Table 5: Complete-request latency and memory at $n = 4096$. Quartiles summarize 15 block means. Logical memory is calculated tensor storage; peak is the framework allocator peak for the whole batch, not total device consumption. One MiB is 2^{20} bytes.

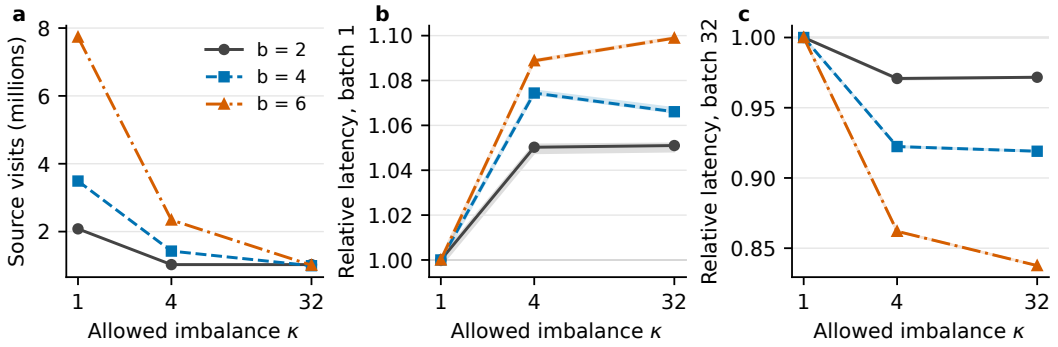


Figure 7: Padding comparison at $n = m = 4096$. For each arity cap b, changing the requested imbalance cap κ changes physical scan padding while preserving the logical graph and horizon. Visit counts include encoding and all physical scan slots. Timing medians and interquartile bands summarize 15 complete-request CUDA Graph block means, divided by the corresponding cap-1 median; these are normalized quartiles, not confidence intervals for a ratio. At $b = 6$, increasing the cap from 1 to 32 removes 87.0% of visits, yet increases batch-1 latency by 9.9%; batch-32 latency falls by 16.2%.

J.3 Workspace and Padding Results

Figure 6 separates the construction counts from elapsed time. At $n = 4096$, the compact loop uses 24 calls and 16,384 visits, compared with 13 calls and 110,592 visits for the full loop. The compact/full latency ratio is 1.651 at batch 1 but 0.4364 at batch 32. Thus the program with fewer visits is slower in the first setting and faster in the second. Table 5 also shows that the compact loop uses less memory at both batch sizes. These are comparisons of the specified implementations; they do not identify the hardware cause of the reversal.

Table 6 gives the exact padding comparison. The calculated effective imbalance is often below the requested cap, and caps 4 and 32 yield identical scan layouts at $b = 2$. At $b = 6$, visits fall from 7,733,248 to 1,009,053 between caps 1 and 32 while the horizon remains 37 calls. The corresponding latency ratios are 1.099 at batch 1 and 0.8376 at batch 32 (Figure 7). Removing padding therefore reduces the number of scan visits, but does not guarantee lower measured latency. Nor does unchanged compact-descriptor memory imply unchanged physical scan work.

J.4 Backend Sensitivity and Reproducibility

Execution policy matters. The implementation launches one update kernel per CoT token and one per binary loop call. CUDA Graphs retain these different numbers of device nodes; a fused or persistent CoT decoder was not tested. The observed CoT/loop timing gap therefore includes the chosen launch structure and cannot be attributed solely to row parallelism. Separate eager-versus-graph diagnostics cover 30 endpoint configurations, using three complete requests per interval on the first input suite. These are diagnostic means, without an estimated interquartile range. For $n = 4096$ at batch 32, eager compact/full times are 0.98965/0.60877 ms, whereas the matched diagnostic graph

Arity cap b	Requested κ	Calls	Visits per example	Effective imbalance
2	1	39	2,080,768	1.0000
2	4	39	1,026,949	2.2872
2	32	39	1,026,949	2.2872
4	1	37	3,489,792	1.0000
4	4	37	1,425,118	2.4869
4	32	37	994,401	3.2849
6	1	37	7,733,248	1.0000
6	4	37	2,341,608	3.3967
6	32	37	1,009,053	11.1224

Table 6: Deterministic resource counts for the capped programs at $n = m = 4096$. All horizons include 32 loading calls. Effective imbalance is the maximum stage ratio of the largest row charge to the average; displayed decimals round exact rational values. Counts are identical per example at both batch sizes.

Backend	Batch	κ	Median ms	Q1–Q3 ms	Peak MiB
Compact CUDA	1	1	3.74202	3.71545–3.75376	1.1235
Materialized reference	1	1	69.6312	69.6025–71.4681	5.4204
Compact CUDA	1	32	3.75709	3.74823–3.78140	1.1235
Materialized reference	1	32	72.3666	72.3246–72.5792	1.5859
Compact CUDA	32	1	3.61643	3.59873–3.75627	16.2700
Materialized reference	32	1	69.2691	68.9217–69.4288	54.9575
Compact CUDA	32	32	3.84726	3.84169–3.85663	16.2700
Materialized reference	32	32	73.0141	70.8202–73.4305	23.5137

Table 7: Separate backend sensitivity at $n = m = 4096$, $b = 6$, under common eager execution. Quartiles summarize five block means per case; peak denotes framework allocator memory for the complete batch. These timings are not pooled with the primary graph measurements.

times are 0.06534/0.11705 ms. Thus even the ranking of the two loops depends on execution policy. The diagnostic graph means are separate from the primary 15-block distributions in Table 5.

Materializing the scans changes the experiment. A separate supplement compares the primary compact CUDA backend with a PyTorch semantic reference that explicitly materializes gathered banks. It fixes $n = m = 4096$, $b = 6$, caps 1 and 32, and both batch sizes, using the same input batch under a common eager policy. Each of the eight cases has ten warmups and five blocks of 20 complete requests. Table 7 reports those distributions. The reference has no paid-instruction preservation claim and its expanded banks and temporary allocations are not attributed to the compact backend. Its substantially larger elapsed times demonstrate sensitivity to the evaluator; they are not a comparison between learned architectures.

Twelve additional materialized-head cases time gather, masking, and segmented extrema together, including allocation and dispatch. They are neither isolated bandwidth measurements nor complete requests. Likewise, GPU thread/SM counts are not the processor parameter in Corollary 22. The experiments justify keeping construction work, dependency depth, and hardware time separate; they do not establish a universal speedup or a causal hardware model.

The source package contains the numerical summaries, timing block records, configuration and provenance, supplementary backend records, and a script rebuilding the paper's empirical figures and main table. The full study archive additionally preserves source snapshots, correctness traces, exact compiled binaries and audits, memory ledgers, and commands for a new run. All reported numbers come from the completed 60-configuration core run and its separately identified supplements; historical development runs are not mixed into the reported results.