EXACT PARSING WITH LOOPED TRANSFORMERS: STORAGE, DEPTH, AND SCHEDULING

Anonymous authors
Paper under double-blind review

ABSTRACT

Exact context-free parsing with looped transformers requires both organizing intermediate states in memory and scheduling updates that advance the computation. We study these requirements through theoretical constructions and experiments on a separate learned parser. On the construction side, we establish storage and loop bounds for strictly causal averaging-hard-attention transformers. For input length n, fixed deterministic and unambiguous context-free languages admit $\mathcal{O}(n^2)$ and $\mathcal{O}(n^{\log_2 7+\varepsilon})$ blank padding, respectively, with $\mathcal{O}(\log^2 n)$ loops including initialization. General context-free languages admit $\mathcal{O}(n^{2\omega_{\mathbb{Q}}+\varepsilon})$ padding, with $\mathcal{O}(\log n)$ loops under supplied fields or $\mathcal{O}(\log^2 n)$ loops including blank initialization. All constructions have fixed parameters, constant width, logarithmic precision, and replayable parsing certificates. On the learning side, learned rankings under a fixed verifier improve completion on some grammars but deteriorate on longer Palindrome inputs. Exploratory scheduling interventions recover most unfinished Palindrome decisions without retraining, while cyclic updates guarantee bounded waiting under the verified-update assumptions. Together, these results identify memory organization and reliable scheduling as complementary means of turning repeated computation into verified progress.

1 INTRODUCTION

Looped transformers apply a shared block repeatedly, allowing the same parameters to support a growing amount of computation (Dehghani et al., 2018; Geiping et al., 2025; Zhu et al., 2025). We investigate this capability through exact context-free parsing, where a fixed set of grammar rules must be composed into larger derivations as input length grows (Aho, 2007, Chapter 4). Checkable derivations make parsing a controlled setting for studying how repeated updates build and verify structured results (Delétang et al., 2022).

Turning this repeated rule application into an exact parser requires both retaining partial results and using them to advance the computation. On the construction side, parallel parsing algorithms reduce sequential depth (Chytil et al., 1991; Jerad et al., 2026), but their intermediate results also require storage. Strictly causal attention can read only earlier token positions, so these results must be arranged to make every required read legal. Auxiliary records and state copies used to achieve this layout can increase storage (Merrill & Sabharwal, 2025a, Lemma 1). The challenge is to preserve the depth reduction while controlling this storage cost. When auxiliary positions start as identical blank tokens, the layout and control fields must also be generated internally. On the learning side, a proposal model must choose which inferences to attempt. A verifier can prevent invalid additions, yet the model may keep overlooking an inference whose prerequisites are already satisfied. Correct updates alone therefore do not ensure completion. These observations lead us to study sufficient resources for exact parsing and scheduling conditions that let learned proposals make progress.

> **RQ1** *What storage and loop depth suffice for exact context-free parsing under strictly causal attention, including initialization?*

Our constructions use interval summaries for deterministic context-free languages (DCFLs), shared arithmetic for unambiguous context-free languages (UCFLs), and classical tree/context composition for general CFLs (Brent & Goldschlager, 1983). We count all auxiliary storage and distinguish

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.

supplied public fields from blank padding initialized within the loop budget. Under the grammar and model conventions of Section 2, the constructed models have fixed parameters, constant width, and logarithmic precision, and produce replayable certificates.

DCFLs admit $\mathcal{O}(n^2)$ blank padding and $\mathcal{O}(\log^2 n)$ loops. For UCFLs, padding is $\mathcal{O}(n^{\omega_{\mathbb{Q}}+\varepsilon})$ with public fields, or $\mathcal{O}(n^{\log_2 7+\varepsilon})$ from blank padding using the rank-seven multiplication identity; both require $\mathcal{O}(\log^2 n)$ loops. General CFLs use $\mathcal{O}(n^{2\omega_{\mathbb{Q}}+\varepsilon})$ padding, with $\mathcal{O}(\log n)$ public-table loops or $\mathcal{O}(\log^2 n)$ loops including blank initialization. Here $\omega_{\mathbb{Q}}$ is the rational matrix-multiplication rank exponent and $\varepsilon > 0$ is arbitrary and fixed. Table 1 summarizes these bounds (Section 3).

> **RQ2** *How does proposal scheduling affect whether verified parsing completes within a finite loop budget?*

We complement these constructions by studying how proposal choices affect completion in a separate learned parser with verified chart updates. Under matched quotas and loop budgets, learned rankings achieve higher mean coverage than Random and Rotating on the Dyck grammars and $a^n b^n$, but deteriorate on longer Palindrome inputs, where enabled updates are persistently missed.

In an exploratory intervention, replacing neural proposals with cyclic scheduling raises the fraction of Palindrome inputs with a verified decision from 18.75% to 97.36% for models trained to exclude verified items, with weights, verifier, quota, and budget unchanged. Cyclic scheduling also guarantees bounded waiting and eventual chart completion under the verified-update assumptions (Section 4).

Implications. These results suggest a broader principle for looped computation: its effectiveness depends on how intermediate results are organized and revisited. Memory layout and scheduling help turn additional iterations into verifiable computational progress.

2 Background and Computational Model

2.1 Languages and Grammars

A context-free grammar (CFG) G generates a context-free language (CFL) $L(G)$. A deterministic CFL (DCFL) is recognized by a deterministic pushdown automaton (DPDA), a stack machine with uniquely determined moves. An unambiguous CFL (UCFL) admits a CFG with exactly one parse tree for each accepted word. These classes satisfy the strict inclusions $\mathrm{DCFL} \subsetneq \mathrm{UCFL} \subsetneq \mathrm{CFL}$ (Autebert et al., 1997). Our constructions fix a DPDA for each DCFL, an unambiguous CFG for each UCFL, and a possibly ambiguous CFG for general CFLs. These automata and grammars do not depend on the input word or its length.

Write $G = (\mathcal{N}, \Sigma, P, S)$ for a fixed CFG, with nonterminals $\mathcal{N}$, terminals Σ, productions P, and start symbol S. Our UCFL and general-CFL constructions use strict Chomsky normal form (CNF), with rules $A \to BC$ or $A \to a$ (Chomsky, 1959), and handle the empty word separately. The UCFL constructions additionally require G to be unambiguous and reduced: every nonterminal occurs in some derivation of a terminal string from S. The chart semantics below allow ambiguity; local updates also allow the acyclic unit rules $A \to B$ used in our examples and experiments.

2.2 Parsing Semantics and Certificates

For an input $w = w_0 \cdots w_{n-1} \in \Sigma^n$, chart parsing records which nonterminals derive which substrings (Aho, 2007, Chapter 4). Its legal item universe is

$$\mathcal{I}_n = \{A[i, j) : A \in \mathcal{N},\ 0 \le i < j \le n\}.$$

A chart is a subset of $\mathcal{I}_n$, whose items $A[i, j)$ assert derivations $A \Rightarrow_G^\star w_i \cdots w_{j-1}$. An item is realizable when this derivation exists; a chart is *sound* when all its items are realizable. A sound chart is *saturated* when it contains all realizable items. For $n > 0$, finding $S[0, n)$ (*root discovery*) in a sound chart certifies acceptance; saturation with the root absent certifies rejection.

Local chart updates. Local parsing starts from lexical matches $A[i, i+1)$ with $A \to w_i$ (Shieber et al., 1995). Each round combines previously known child items using binary rules. The binary

consequences of a chart X are

$$D(X) = \{A[i,j) : \exists (A \to BC) \in P,\ \exists k :\ i < k < j,\ B[i,k), C[k,j) \in X\}.$$

The unit closure $\mathrm{unitcl}(X)$ repeatedly adds $A[i,j)$ to X whenever $A \to B$ and $B[i,j)$ is present, until stable. Let $B(w)$ be the unit closure of the lexical matches. Starting from $I_0 = B(w)$, apply $I_{t+1} = \mathrm{unitcl}(I_t \cup D(I_t))$. This reaches the least fixed point containing $B(w)$, denoted $I^\star$, the exact chart. Each exhaustive update is one local *semantic round*.

Reachability corrections. A correction follows an entire dependency chain with known side branches, rather than advancing just one binary level. For a unit-free strict-CNF grammar, fix a sound chart X and form a graph on all legal items $\mathcal{I}_n$. For each rule $A \to BC$ and split $i < k < j$, add $A[i,j) \to B[i,k)$ if $C[k,j) \in X$, and $A[i,j) \to C[k,j)$ if $B[i,k) \in X$. Proving the indicated child then proves its parent. Keep X and the graph fixed during the correction; a path ending in X proves every item along it. Let $\mathcal{C}(X)$ consist of X and all items reaching X in this graph. Starting from $X^{(0)} = B(w)$, iterate $X^{(t+1)} = \mathcal{C}(X^{(t)})$. This is Newton iteration in Boolean chart form (Esparza et al., 2010, Proposition 7.1(b)). Section A relates these rounds to derivation structure.

Parsing certificates. A parsing certificate stores one valid rule for each certified item, with child references and a split for binary rules. Following these references to lexical matches verifies the item and, from $S[0,n)$, recovers a parse tree. DCFL certificates instead record accepting DPDA computations; fixed annotations recover derivations for an associated grammar.

2.3 Attention model

For query, key, and value vectors q_u, k_u, v_u at input or auxiliary position u and a nonempty mask $\mathcal{M}_u$, an averaging-hard-attention (AHAT) head (Merrill & Sabharwal, 2025a) returns

$$h_u = \frac{1}{|\mathcal{A}_u|} \sum_{s \in \mathcal{A}_u} v_s, \qquad \mathcal{A}_u = \arg\max_{s \in \mathcal{M}_u} \langle q_u, k_s \rangle. \tag{1}$$

AHAT averages the values of all tied maximizers among the positions in $\mathcal{M}_u$; if $\mathcal{M}_u = \varnothing$, the head returns zero. The strictly causal constructions in Section 3 use $\mathcal{M}_u = \{v : v < u\}$: every attention read accesses an earlier physical position, while residual updates retain the token's own state.

Each loop applies the same constant-depth block of attention heads, pointwise operations, and residual updates. The block and its parameters are fixed across loops and input lengths; width and head count are constant. Head and pointwise maps use constantly many independently normalized rational projections, followed by fixed linear or ReLU maps, under the multi-pre-norm convention of Jerad et al. (2026). Each scalar uses $\mathcal{O}(\log n)$ bits. Numerical conventions and finite-precision retrieval guarantees are given in Sections B.2, B.8.1 and B.8.4.

2.4 Storage and loop budgets

For input length n, $M(n)$ counts all input and auxiliary token positions, including lookup tables, reserved positions, and retained state copies. Reusing a position does not increase $M(n)$; allocating a new copy does. Each record holds a constant number of $\mathcal{O}(\log n)$-bit fields. Let $L(n)$ count applications of the shared transformer block.

The *public-table model* supplies addresses, layout tags, and numerical tables generated by a fixed logspace procedure from n and the construction, without reading input symbols. These records count toward $M(n)$, but $L(n)$ excludes their external generation (Section B.9). The *blank-padding model* starts with a fixed beginning-of-sequence (BoS) marker, input-symbol embeddings, and identical blank tokens, in that order, with no supplied positional fields. The model generates its fields internally; $L(n)$ includes initialization and subsequent parsing (Section B.8).

3 Causal constructions for exact parsing

Implementing parallel parsing with strictly causal attention requires arranging intermediate states so that every read accesses an earlier physical position. A separate copy of the full state for each

simulated layer provides this order, but multiplies storage by the simulation depth. Our constructions control total storage by exploiting the structure of parsing dependencies. For the blank-padding bounds, the required layouts and schedules are also generated internally.

Our three constructions address these costs differently. DCFL parsing merges interval summaries in reusable work banks. UCFL parsing implements reachability corrections with shared arithmetic circuits to reduce storage. Both use unique-path dependency graphs and status contraction. For general CFLs, tree/context composition handles ambiguity without requiring unique paths. Table 1 compares the resulting bounds. All constructions use logarithmic precision and the initialization conventions of Section 2.4. Here $\omega_{\mathbb{Q}} = \inf_{m\geq 2} \log_m \operatorname{rank}_{\mathbb{Q}}(\mathrm{MM}_m)$ is the rational rank exponent of matrix multiplication, where MM_m is the tensor for $m \times m$ matrix multiplication.

3.1 Deterministic parsing by reusable interval banks

Theorem 3.1 (AHAT construction for DCFLs)**.** *For every DCFL $\mathcal{L}$, there exists a fixed-parameter, constant-width, strictly causal looped AHAT that decides membership in $\mathcal{L}$ on every length-n input using $\mathcal{O}(n^2)$ blank padding, $\mathcal{O}(\log n)$-bit precision, and $\mathcal{O}(\log^2 n)$ loops, including initialization. On accepted inputs, it retains a replayable certificate of an accepting computation of a fixed DPDA for $\mathcal{L}$, annotated to recover a derivation in an associated fixed CFG generating $\mathcal{L}$.*

Call summaries and interval merging. Adapting Chytil et al. (1991), we summarize stack calls of a fixed normalized DPDA. A summary maps a surface configuration $c = (q, i, A)$—state, input position, and stack-top symbol—to the surface configuration $\tau(c)$ immediately before that stack frame is popped, if the call returns. The stack below the frame is untouched until the return, so the summary need not record it. We recursively bisect the input interval, using immediate returns and one-symbol calls as base cases, and merge the summaries of the two halves. Direct subcalls execute sequentially and have disjoint interval interiors, so at most one crosses the midpoint. Each possible call return defines a candidate summary. After looking up known subcalls, dependency edges link parent candidates to candidate returns of the missing child. Known summaries and those computed entirely from known subcalls form the verified bases. A candidate is valid exactly when it reaches a verified base. The accepting summary decides membership (Section B.5).

Unique-path contraction and causal reuse. Normalization makes every nontrivial call consume input before its subcalls, ensuring acyclicity. The start of the first missing subcall depends only on the parent start and known summaries; each candidate child return configuration determines a unique parent return configuration. These properties give at most one path between any two candidate vertices, including false candidates (Lemma B.3). We apply the status-contraction procedure of Lemma B.1: it propagates resolved candidate values and replaces a sole remaining unresolved dependency by a pointer. Pointer doubling compresses chains of these dependencies, resolving the $\mathcal{O}(m^2)$ candidates of a length-m interval in $\mathcal{O}(\log m)$ rounds. For causal execution, completed child-interval outputs precede the parent's work bank. Within that bank, decreasing input starts give the order *child candidate, edge helper, parent candidate*. Helpers relay child-candidate statuses to their parents, implementing the required status queries in a constant number of AHAT stages (Section B.5 and Proposition B.5). Composed pointers still target earlier records, so each merge reuses its work positions across rounds while retaining completed outputs (Section B.2.3).

Resource bounds, initialization, and certificates. With the automaton fixed, $\mathcal{O}(m)$ starts and $\mathcal{O}(m)$ terminators give $\mathcal{O}(m^2)$ candidate summaries per length-m interval. Auxiliary records fit within the same bound. Storage sums over all interval banks; execution depth $D(n)$ follows one recursion path because the halves run in parallel. For power-of-two lengths,

$$M(n) = \sum_{r=0}^{\log_2 n} 2^r \mathcal{O}\big((n/2^r)^2\big) = \mathcal{O}(n^2), \qquad D(n) = D(n/2) + \mathcal{O}(\log n) = \mathcal{O}(\log^2 n). \tag{2}$$

Power-of-two padding preserves these bounds. From blank padding, Proposition B.5 initializes the layout and schedule of all interval banks in parallel in $L_{\mathrm{init}}(n) = \mathcal{O}(\log^2 n)$ loops within the same quadratic storage bound. Thus total depth is $L(n) = L_{\mathrm{init}}(n) + D(n) = \mathcal{O}(\log^2 n)$. Witness fields retain selected original edges while working pointers are compressed. Following these edges and

Table 1: **Parsing constructions compared by language class.** All rows have constant width and logarithmic precision. Witness indicates a retained replayable parsing certificate within the listed bounds; "—" means no such guarantee is specified. Initialization and resource accounting follow Section 2.4. In the Model column, AHAT denotes strictly causal attention; MAHAT mixes causal and unmasked heads. Rows marked "+" apply the full-layer causal simulation of Merrill & Sabharwal (2025a) to the cited source construction. For DCFLs, the source is our public-field AHAT realization of the Chytil et al. (1991) framework. For UCFLs and general CFLs, the sources are the mixed-mask constructions of Jerad et al. (2026).

Grammar	Construction	Model	Padding	Loops	Initialization	Witness
DCFL	Chytil et al. (1991) + Merrill (2025a)	AHAT	$\mathcal{O}(n^2 \log^2 n)$	$\mathcal{O}(\log^2 n)$	Public fields	Yes
	Ours (Thm. 3.1)	AHAT	$\mathcal{O}(n^2)$	$\mathcal{O}(\log^2 n)$	Blank padding	Yes
UCFL	Jerad et al. (2026)	MAHAT	$\mathcal{O}(n^3)$	$\mathcal{O}(\log^2 n)$	Blank padding	—
	Jerad et al. (2026)	AHAT	$\mathcal{O}(n^4)$	$\mathcal{O}(\log^2 n)$	Blank padding	—
	Jerad et al. (2026) + Merrill (2025a)	AHAT	$\mathcal{O}(n^3 \log^2 n)$	$\mathcal{O}(\log^2 n)$	Blank padding	—
	Ours (Thm. 3.2(i))	AHAT	$\mathcal{O}(n^{\omega_{\mathbb{Q}}+\varepsilon})$	$\mathcal{O}(\log^2 n)$	Public fields	Yes
	Ours (Thm. 3.2(ii))	AHAT	$\mathcal{O}(n^{\log_2 7+\varepsilon})$	$\mathcal{O}(\log^2 n)$	Blank padding	Yes
General CFL	Jerad et al. (2026)	MAHAT	$\mathcal{O}(n^6)$	$\mathcal{O}(\log n)$	Blank padding	—
	Jerad et al. (2026)	AHAT	$\mathcal{O}(n^7)$	$\mathcal{O}(\log n)$	Blank padding	—
	Jerad et al. (2026) + Merrill (2025a)	AHAT	$\mathcal{O}(n^6 \log n)$	$\mathcal{O}(\log n)$	Blank padding	—
	Ours (Thm. 3.3)	AHAT	$\mathcal{O}(n^{2\omega_{\mathbb{Q}}+\varepsilon})$	$\mathcal{O}(\log n)$	Public fields	Yes
	Ours (Thm. 3.3)	AHAT	$\mathcal{O}(n^{2\omega_{\mathbb{Q}}+\varepsilon})$	$\mathcal{O}(\log^2 n)$	Blank padding	Yes

saved transitions from the accepting summary to immediate-return bases recovers the accepting computation and, with fixed annotations, a derivation in the associated grammar (Section B.5).

3.2 Unambiguous parsing by shared arithmetic circuits

Theorem 3.2 (AHAT construction for UCFLs)**.** *For every fixed reduced unambiguous grammar G in strict CNF and every fixed $\varepsilon > 0$, there exist fixed-parameter, constant-width, strictly causal looped AHATs that, on every length-n input, construct the exact chart and retain a replayable parsing certificate for each realizable item. The padding bounds are:*

(i) ***Public initialization:*** *$\mathcal{O}(n^{\omega_{\mathbb{Q}}+\varepsilon})$ padding.*
(ii) ***Blank initialization:*** *$\mathcal{O}(n^{\log_2 7+\varepsilon})$ padding.*[1]

Both use $\mathcal{O}(\log n)$-bit precision and $\mathcal{O}(\log^2 n)$ loops. The loop bounds exclude external generation in (i) *and include internal initialization in* (ii)*.*

Corrections and unique-path contraction. We evaluate each reachability correction of Section 2.2 with $X \cup D(X)$ as verified bases; this preserves $\mathcal{C}(X)$ since $D(X)$ reaches X in one step. Each edge shortens the span, making the graph acyclic. Reducedness and unambiguity give the unique-path property, even for unrealizable candidates: two distinct paths with the same endpoints give two partial derivations with the same hole nonterminal. Filling both holes with the same terminal derivation in a common start context gives two parses of one word (Section B.6.1). Statuses and pointers are reinitialized for each correction, so a false status applies only to that correction. With $\mathcal{O}(n^2)$ vertices, Lemma B.1 gives $\mathcal{O}(\log n)$ contraction rounds per correction. From lexical items, Proposition A.2 gives $\mathcal{O}(\log n)$ corrections for the full chart. The constant-depth queries and pointer updates below thus yield $\mathcal{O}(\log^2 n)$ loops overall.

Shared arithmetic and causal layout. Each contraction round needs dependency counts and the address of a sole unresolved child. For each rule, orientation, and queried status, Boolean matrices L, R encode the child's status and its sibling's membership in X. Matrix products give the counts

[1] Part (ii) uses Strassen's rank-seven matrix-multiplication identity (Strassen, 1970); both parts apply to every grammar in the statement.

and split-weighted counts:

$$c_{ij} = \sum_{i<k<j} L_{ik} R_{kj}, \qquad d_{ij} = \sum_{i<k<j} k L_{ik} R_{kj}. \tag{3}$$

Aggregating counts detects true dependencies and distinguishes zero, one, or multiple unresolved dependencies. When the unresolved total is one, the nonzero channel identifies the rule and orientation, and d_{ij} gives the split, determining the child address (Eq. (21)). For public initialization, fast bilinear matrix multiplication shares intermediate linear forms, giving subcubic circuits of size $\mathcal{O}(n^{\rho})$ for a fixed rational identity with exponent $2 < \rho < 3$. Expanding blocks of linear operations into parallel relays trades additional records for constant depth. Normalized Boolean-gate circuits recover exact integer outputs as bits at logarithmic precision. For a fixed compression parameter h, each product uses $\mathcal{O}(n^{\rho+1/h})$ records and $\mathcal{O}(h)$ stages (Section B.4.3). Successive rounds place old snapshots, arithmetic computations, and status and pointer updates in fresh banks in dependency order, making all reads strictly causal. Choose the identity and h, independently of n, with $\rho + 1/h < \omega_{\mathbb{Q}} + \varepsilon$. Then

$$M(n) = \mathcal{O}\Big(n^{\rho+1/h} \log^2 n\Big) = \mathcal{O}(n^{\omega_{\mathbb{Q}}+\varepsilon}). \tag{4}$$

The exponent margin absorbs the stage copies. Constant-depth queries and pointer updates give $L(n) = \mathcal{O}(\log^2 n)$ execution loops (Section B.6.2).

Blank initialization and parsing certificates. From blank padding, the model initializes record roles, connections, and control fields for the rank-seven circuit, using local arithmetic circuits to implement the exact count and weighted-count interface in Eq. (3). Canonical keys preserve exact tied aggregation, synchronization ensures input readiness, and controlled numerical error preserves discrete outputs at logarithmic precision (Lemmas B.19 and B.20). With a sufficiently large fixed compression parameter, the additional arithmetic and stage-copy costs fit within $\mathcal{O}(n^{\log_2 7+\varepsilon})$ records; initialization and execution together take $\mathcal{O}(\log^2 n)$ loops (Proposition B.9). For either initialization, reducedness and unambiguity give a unique successful rule and split for each realizable nonlexical item (Lemma B.6). Setting both factors in Eq. (3) to child-item indicators from the completed chart identifies the successful rule and recovers its split. An $\mathcal{O}(n^2)$ output bank stores the rule and child references; child spans strictly decrease, so replay terminates at lexical items (Section B.6.2 and Proposition B.9).

3.3 General parsing by tree–context composition

Theorem 3.3 (AHAT construction for general CFLs)**.** *For every fixed grammar G in strict CNF and every fixed $\varepsilon > 0$, there exist fixed-parameter, constant-width, strictly causal looped AHATs that, on every length-n input, construct the exact chart and retain a replayable parsing certificate for each realizable item. These constructions use $\mathcal{O}(n^{2\omega_{\mathbb{Q}}+\varepsilon})$ padding and $\mathcal{O}(\log n)$-bit precision. With public initialization, they require $\mathcal{O}(\log n)$ loops, excluding external generation; with blank initialization, they require $\mathcal{O}(\log^2 n)$ loops, including internal initialization.*

Trees, contexts, and logarithmic convergence. With ambiguity, dependency graphs need not have unique paths. We use classical tree/context doubling (Brent & Goldschlager, 1983): the Boolean vector $P_t[u]$ records whether item u has an established complete derivation, and $Q_t[u, z]$ records whether a one-hole context from outer item u to hole item z has been established. Such a context is a partial parse whose only unexpanded subtree is the hole; no derivation of the hole's specified input substring is required. Initialize P_0 to the lexical chart and Q_0 to the identity matrix, representing contexts consisting only of their hole. Each phase wraps contexts with known sibling trees, composes two contexts, and fills holes with known trees or immediate binary consequences (Eq. (24)). A simultaneous induction shows that each phase doubles the size thresholds for covering all valid trees and contexts. Splitting a context at a suitable node on its hole path gives two smaller contexts and a sibling tree; a complete tree splits into an outer context and a subtree formed from two known children. These operations reconstruct each structure from the smaller pieces. Since strict-CNF trees and contexts on length-n inputs have $\mathcal{O}(n)$ nodes, $\mathcal{O}(\log n)$ phases complete the chart (Lemma B.10).

Boolean products and causal storage. Context composition is a Boolean matrix product: it tests whether an intermediate item connects an outer context to an inner one. Since each matrix index is an

item, the matrix side is $v = |\mathcal{I}_n| = \Theta_G(n^2)$. For public initialization, the shared arithmetic and depth compression of Section B.4.3 implement each product in constant depth using $\mathcal{O}(v^{\omega_{\mathbb{Q}}+\delta})$ records, for any fixed $\delta > 0$. The arithmetic product counts matching intermediate items; testing its entries for positivity recovers the Boolean relation. Signed intermediate values are retained until the product is complete. Each doubling phase uses a constant number of these products. Fresh banks place old relations, arithmetic computations, and new relations in dependency order, so all reads access earlier records. Choosing $2\delta < \varepsilon$ gives

$$M(n) = \mathcal{O}\big(v^{\omega_{\mathbb{Q}}+\delta} \log n\big) = \mathcal{O}\big(n^{2\omega_{\mathbb{Q}}+2\delta} \log n\big) = \mathcal{O}(n^{2\omega_{\mathbb{Q}}+\varepsilon}). \tag{5}$$

The remaining exponent margin absorbs the phase copies. Constant-depth phases give $L(n) = \mathcal{O}(\log n)$ public execution loops at logarithmic precision (Proposition B.11).

Blank initialization and witness selection. For Boolean products, only the positivity of each completed entry is needed. Clearing denominators of any fixed rational multiplication identity introduces a positive common scale, so the positivity test needs no recovery of unscaled values. Signed intermediates are retained until the final threshold (Eq. (25)). Lemma B.12 internally initializes these product modules at logarithmic precision while preserving the storage bound in Eq. (5). The static fields of all phase banks initialize once, in parallel, before parsing begins; initialization takes $\mathcal{O}(\log^2 n)$ loops and subsequent parsing takes $\mathcal{O}(\log n)$, giving $\mathcal{O}(\log^2 n)$ loops in total (Section B.7.3). For either initialization, an additional $\mathcal{O}_G(n^3)$ bank tests candidate rules and splits against the completed chart. A fixed priority selects one successful candidate per realizable nonlexical item; lexical items retain their terminal rule. Storing the selected rule and child references gives a replayable certificate, since child spans strictly decrease. This bank fits within the stated storage bound (Proposition B.11 and Section B.7.3).

4 Proposal Scheduling and Verified Completion

The constructions above establish sufficient resources for exact parsing. We now study RQ2 in a learned parser: how does proposal scheduling affect completion within a finite loop budget? We compare proposal policies, diagnose failures on longer inputs, and test whether scheduling changes restore progress without retraining.

4.1 Verified Updates and Evaluation

The parser starts from the unit closure of lexical items. Each loop, a policy proposes items up to its quota; the verifier adds those whose binary-rule children were verified before this update, recording their rule and split. Existing items are retained; unit closure is applied outside the quota. Neural policies rank items, not derivations. Missing items enabled by binary rules form the *frontier*.

A verified root item $S[0, n)$ certifies acceptance; a chart closed under all grammar rules with the root absent certifies rejection. Inputs that reach neither condition within the loop budget remain unresolved (Proposition D.1). We measure completion by sound-decision coverage:

$$\text{Coverage}(L) = \frac{\#\{\text{inputs accepted or rejected by loop } L\}}{\#\{\text{evaluation inputs}\}}. \tag{6}$$

We compare policies at the same per-loop proposal limit (quota) and loop budget on five grammars: Dyck(1), Dyck(2), $a^n b^n$, Palindrome, and Arithmetic. Out-of-distribution (OOD) evaluation uses inputs longer than those used for training. Across three paired seeds, training policies share data and initialization within each pair. The verifier computes all binary consequences before filtering by proposals, so the comparison measures finite-budget completion rather than end-to-end runtime or work. Hyperparameters are listed in Table 3; data, architecture, and training details are in Section D.2.

4.2 Eligibility and Proposal Order

We first study eligibility at two stages: training and inference. At either stage, policy A allows all legal typed spans, while N excludes verified items without changing the quota. Training under each policy and evaluating each fixed checkpoint under both gives four combinations: AA, AN, NA,

Table 2: **Effects of eligibility and proposal order on OOD completion.** Three paired seeds. **(a)** Absolute coverage differences (%), averaged across five grammars, with unadjusted 95% CIs; interaction is $(NN - AN) - (NA - AA)$. **(b)** Mean coverage (%) across seeds with N inference, matched quotas, and 120 loops (240 for Arithmetic). Random averages three schedule repeats per input before seed aggregation; Oracle denotes the frontier oracle using privileged verifier information.

(a) Eligibility contrasts

Contrast	Δ (%)	95% CI
$AN - AA$	+4.14	[+1.58, +6.69]
$NN - NA$	+2.53	[+0.87, +4.19]
$NN - AN$	−2.78	[−9.65, +4.10]
$NA - AA$	−1.17	[−4.59, +2.25]
Interaction	−1.61	[−5.78, +2.56]

(b) Decision coverage (%)

Grammar	AN	NN	Static	Rotating	Random	Oracle
Dyck(1)	93.89	92.78	0.00	79.58	65.00	100.00
Dyck(2)	97.36	97.50	0.14	84.03	71.94	100.00
$a^n b^n$	90.56	89.03	0.00	58.06	36.16	100.00
Palindrome	30.00	18.75	0.00	**99.31**	76.44	100.00
Arithmetic	99.86	99.72	0.00	100.00	100.00	100.00

and NN, with training first and inference second. Table 2(a) shows that switching inference to N improves five-grammar mean coverage under either training policy. With N inference fixed, training with N shows no clear additional benefit (Section D.3).

With N inference fixed, we next compare proposal orders. Alongside neural AN and NN, we compare four orders: Static restarts each scan at the same address, while Rotating advances its start cyclically; Random samples unverified items uniformly each loop; Frontier oracle prioritizes enabled items using verifier information. All use the same quota and loop budget (Section D.3). In Table 2(b), AN and NN have higher mean coverage than Random and Rotating on Dyck and $a^n b^n$, but trail both on Palindrome; Arithmetic is nearly saturated except under Static.

4.3 Ranking failures on longer inputs

Palindrome is the clearest failure case: many inputs remain unresolved under neural scheduling after 120 loops (Fig. 1, left). To diagnose this failure, we measure the fraction of currently enabled missing items selected at each update. Under NN, this fraction falls from 98.8% in distribution to 8.1% and 5.6% in the 25–48 and 49–60 token OOD groups; AN also deteriorates (Fig. 1, middle). Together with the rank and stall diagnostics, these omissions suggest *proposal starvation* on longer inputs: enabled updates are repeatedly missed (Section D.4). At loop 120, the oldest enabled item on unresolved inputs has waited a median 118 loops under either training policy. We next test whether changing only the schedule restores progress.

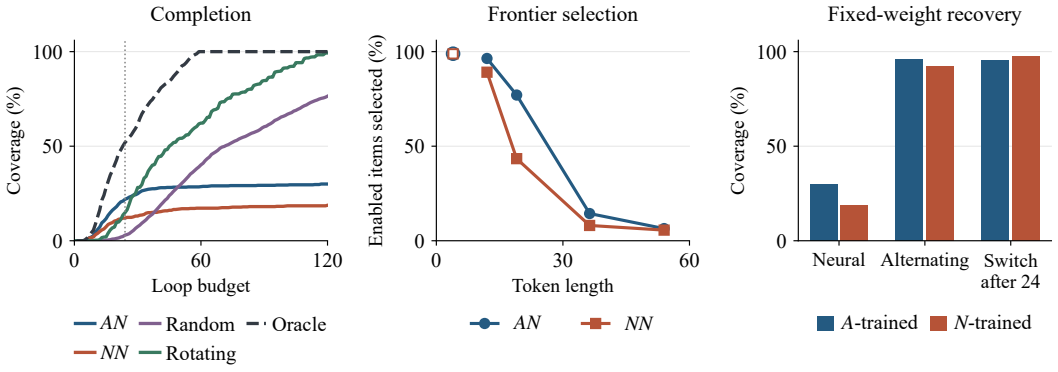


Figure 1: **Palindrome: failure, diagnosis, and recovery. Left:** OOD coverage versus loop budget, averaged over three seeds; the dotted line marks the training horizon. **Middle:** Mean fraction of enabled missing items selected, by token length. Hollow points are in-distribution (ID) references with training-word overlap; filled points are OOD. Only unresolved updates with a nonempty frontier contribute; averaging details are in Section D.4. **Right:** Coverage after 120 loops under fixed-weight scheduling interventions, for both training policies with N inference. Quotas are matched per input.

4.4 SCHEDULING INTERVENTIONS AND BOUNDED WAITING

We change the proposal schedule on Palindrome while fixing model weights, verifier, quota, and the 120-loop budget. Under N inference, Alternating uses neural proposals on odd loops and Rotating on even loops; Switch after 24 uses neural proposals for 24 loops, then Rotating. With N-trained weights, coverage rises from 18.75% to 92.22% and 97.36%, respectively; both interventions also recover most unfinished decisions with A-trained weights (Fig. 1, right). These exploratory interventions show that scheduling can restore progress; execution details are in Section D.5.

Bounded waiting. Cyclic scanning prevents enabled updates from waiting indefinitely. Under the verified-update protocol of Section 4.1, let m count all legal items, including verified ones, and let $1 \le q \le m$ be the quota. Every pth loop, Rotating selects the first q unverified items in cyclic address order (or all if fewer remain) and advances its starting address by q. Within $\lceil m/q \rceil$ such scans, every address is covered, so each continuously enabled missing item is added within B loops:

$$B = p\lceil m/q \rceil, \qquad \text{loops to full chart closure} \le BH. \tag{7}$$

Here H counts exhaustive updates from lexical and unit initialization to closure. Since enabled items remain enabled, each such update takes at most B loops (Proposition D.2 and Section D.6).

5 RELATED WORK

Looped transformers and exact computation. Recurrent attention and adaptive computation support repeated updates (Dehghani et al., 2018; Graves, 2016). Expressivity results connect loop depth to parallel computation: logarithmic depth suffices for regular languages and graph connectivity (Merrill & Sabharwal, 2025b), while padded AHATs admit circuit-class characterizations (Merrill & Sabharwal, 2025a) whose architectural and precision assumptions are examined by Svete et al. (2026). For context-free languages, Jerad et al. (2026) implement parallel recognition with smaller padding in the unambiguous case. We give storage bounds for strictly causal parsing constructions, including internal initialization from blank padding.

Parallel parsing and derivation structure. Parallel parsing exploits both derivation structure and shared arithmetic. Unique-path graph evaluation (Chytil et al., 1991) and strip decompositions (Larmore & Rytter, 1998) organize parallel evaluation, while derivation dimension bounds correction rounds (Esparza et al., 2010). Algebraic recognition reduces parsing to matrix multiplication (Valiant, 1974), allowing bilinear identities to share intermediate computations (Strassen, 1970). Tree/context composition combines these approaches for general CFLs, with a processor exponent twice the matrix-multiplication exponent (Brent & Goldschlager, 1983). We implement these methods through reusable DCFL banks, shared UCFL arithmetic, and causal tree/context layouts.

Learning and scheduling for context-free parsing. One approach to learning context-free structure equips models with pushdown memory, from differentiable stacks (Joulin & Mikolov, 2015; Grefenstette et al., 2015) and nondeterministic stacks (DuSell & Chiang, 2020) to transformer stack attention (DuSell & Chiang, 2024). Such memory choices affect generalization to longer inputs (Delétang et al., 2022). Alongside memory design, learned agenda priorities address which parsing actions to execute, trading accuracy against inference cost (Jiang et al., 2012). We study finite-budget completion under a fixed verifier and prove bounded waiting for cyclic scheduling.

6 CONCLUSION

We studied how memory organization and proposal scheduling support exact parsing with looped transformers. Our strictly causal constructions retain replayable witnesses and provide explicit storage, depth, and initialization bounds for deterministic, unambiguous, and general context-free languages. In a separate learned parser, proposal rankings improve finite-budget completion on some grammars but deteriorate on longer Palindrome inputs. Scheduling interventions recover most unfinished Palindrome decisions without retraining, while cyclic updates guarantee bounded waiting under the verified-update assumptions. Together, these results highlight memory layout and reliable scheduling as complementary ways to turn repeated computation into verified completion.

REPRODUCIBILITY STATEMENT

Section D specifies the grammars, training, evaluation, paired seeds, and statistical comparisons. Sections D.3, D.4, D.6 and D.7 report scheduling controls, diagnostics, and the DVPA study. Supplementary code and per-run results support independent reproduction; finite component checks complement the all-length proofs in the appendices.

ETHICS STATEMENT

All experiments use synthetic strings generated from formal grammars and serialized automata. No human participants, personal data, or sensitive information are involved at any stage of this study.

AI USE STATEMENT

Generative AI assisted with manuscript editing, code development, organization of experiment records, image generation, consistency checks, and assistance with proof revision. The authors are responsible for the proofs, claims, code, and reported results.

REFERENCES

Alfred V Aho. *Compilers: Principles, Techniques and Tools (for VTU)*. Pearson Education India, 2007.

Rajeev Alur and Parthasarathy Madhusudan. Visibly pushdown languages. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pp. 202–211, 2004.

Jean-Michel Autebert, Jean Berstel, and Luc Boasson. Context-free languages and pushdown automata. In *Handbook of Formal Languages: Volume 1 Word, Language, Grammar*, pp. 111–174. Springer, 1997.

Dario Bini. Relations between exact and approximate bilinear algorithms. applications. *Calcolo*, 17 (1):87–97, 1980.

Richard Peirce Brent and Leslie Goldschlager. *A parallel algorithm for context-free parsing*. Australian National University, Centre for Mathematical Analysis, 1983.

Noam Chomsky. On certain formal properties of grammars. *Information and control*, 2(2):137–167, 1959.

Michal Chytil, Maxime Crochemore, Burkhard Monien, and Wojciech Rytter. On the parallel recognition of unambiguous context-free languages. *Theoretical Computer Science*, 81(2):311–316, 1991.

Thomas M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE transactions on electronic computers*, (3):326–334, 1965.

Nicolaas Govert De Bruijn. A combinatorial problem. In *Nederl. Akad. Wetensh. Proc.*, volume 49, pp. 758–764, 1946.

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

Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, et al. Neural networks and the chomsky hierarchy. *arXiv preprint arXiv:2207.02098*, 2022.

Brian DuSell and David Chiang. Learning context-free languages with nondeterministic stack rnns. In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pp. 507–519, 2020.

Brian DuSell and David Chiang. Stack attention: Improving the ability of transformers to model hierarchical patterns. In *International Conference on Learning Representations*, volume 2024, pp. 7411–7430, 2024.

Javier Esparza, Stefan Kiefer, and Michael Luttenberger. Newtonian program analysis. *Journal of the ACM (JACM)*, 57(6):1–47, 2010.

Jürgen Forster. A linear lower bound on the unbounded error probabilistic communication complexity. *Journal of Computer and System Sciences*, 65(4):612–625, 2002.

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. *Advances in Neural Information Processing Systems*, 38: 41340–41391, 2025.

Robert Glück. Simulation of two-way pushdown automata revisited. *arXiv preprint arXiv:1309.5142*, 2013.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. *Advances in neural information processing systems*, 28, 2015.

William Hesse, Eric Allender, and David A Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65(4): 695–716, 2002.

Selim Jerad, Anej Svete, Sophie Hao, Ryan Cotterell, and William Merrill. Context-free recognition with transformers. *arXiv preprint arXiv:2601.01754*, 2026.

Jiarong Jiang, Adam Teichert, Jason Eisner, and Hal Daume. Learned prioritization for trading off accuracy and speed. *Advances in Neural Information Processing Systems*, 25, 2012.

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. *Advances in neural information processing systems*, 28, 2015.

Joseph M Landsberg and Giorgio Ottaviani. New lower bounds for the border rank of matrix multiplication. *Theory of Computing*, 11(1):285–298, 2015.

Lawrence L Larmore and Wojciech Rytter. Almost optimal sublinear time parallel recognition algorithms for three subclasses of context free languages. *Theoretical computer science*, 197(1-2): 189–201, 1998.

Will Merrill and Ashish Sabharwal. Exact expressive power of transformers with padding. *Advances in Neural Information Processing Systems*, 38:112497–112524, 2025a.

Will Merrill and Ashish Sabharwal. A little depth goes a long way: The expressive power of log-depth transformers. *Advances in Neural Information Processing Systems*, 38:95315–95339, 2025b.

Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1): 145–147, 1972.

Stuart M Shieber, Yves Schabes, and Fernando CN Pereira. Principles and implementation of deductive parsing. *The Journal of logic programming*, 24(1-2):3–36, 1995.

Volker Strassen. Gaussian elimination is not optimal. *Matematika*, 14(3):127–128, 1970.

Anej Svete, William Merrill, Ryan Cotterell, and Ashish Sabharwal. Revisiting padded transformer expressivity: Which architectural choices matter and which don't. *arXiv preprint arXiv:2605.30523*, 2026.

Leslie Valiant. General context-free recognition in less than cubic time. 1974.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A SEMANTIC ROUNDS AND DERIVATION STRUCTURE

Using the chart semantics of Section 2.2, we relate local rounds to binary derivation height and correction rounds to Strahler dimension (Propositions A.1 and A.2). Both results allow ambiguity: local updates permit acyclic unit rules, while corrections use unit-free strict CNF. These characterizations adapt classical fixed-point analysis (Esparza et al., 2010, Proposition 6.4, Theorem 6.8, and Proposition 7.1(b)). Here, a round is one exhaustive semantic update; transformer loop costs and proposal-limited runs are treated in Sections 3 and 4.

A.1 LOCAL PROPAGATION MEASURES BINARY HEIGHT

Starting from the lexical bases and their unit closure, local propagation applies

$$I_0 = B(w), \qquad I_{t+1} = \mathrm{unitcl}\big(I_t \cup D(I_t)\big). \tag{8}$$

The local root-discovery time is $T^\star(w) = \min\{t \geq 0 : S[0,n) \in I_t\}$, with $\min \varnothing = \infty$. The binary height $h(T)$ counts the maximum number of binary rules on a root-to-leaf path: lexical trees have height zero, unit nodes preserve height, and binary nodes have height $1 + \max(h_L, h_R)$.

Proposition A.1 (Local rounds and binary height)**.** *For a CFG with lexical and binary rules and optional acyclic unit rules, I_t contains an item exactly when it has a derivation of binary height at most t, for every $t \geq 0$. In particular, for $w \in L(G)$, $T^\star(w) = \min_{T:S\Rightarrow^\star w} h(T)$.*

Proof of Proposition A.1. At $t = 0$, the initialized items have lexical derivations and any acyclic unit-chain extensions, exactly the trees of binary height zero. Assume the claim at round t. Retained items admit trees of height at most t; a binary addition combines two trees of height at most t, giving height at most $t + 1$. Unit closure preserves height. Conversely, remove any unit chain at the root of a tree of height at most $t + 1$. The remaining root is lexical, or binary with two child trees of height at most t. Lexical items are already present; in the binary case, the induction hypothesis makes both children available in round t. The next update derives the parent, and unit closure restores the removed chain. Taking the earliest root round gives the minimum height. □

Root discovery and full closure. The itemwise characterization also gives the exhaustive-update count needed to saturate the chart:

$$H(w) := \min\{t \geq 0 : I_t = I^\star\} = \max_{u \in I^\star} \min_{T \text{ derives } u} h(T), \tag{9}$$

with the maximum defined as zero if $I^\star$ is empty. On accepted inputs, $T^\star(w) \leq H(w)$: the root may be found before the chart is saturated. This $H(w)$ is the closure count in the BH guarantee of Section 4.4 and Proposition D.2; the corresponding acceptance and rejection heights are measured in Section D.6.

Dyck grammar and local-height calculation. Consider the following grammar for nonempty balanced words with one bracket type, with start symbol D:

$$D \to PD \mid P, \quad P \to LR \mid LX, \quad X \to DR, \quad L \to (, \quad R \to).$$

Here D concatenates balanced blocks P, and X splits the wrapping operation into binary rules. Write $h(w)$ for the minimum binary height of a derivation of w from D. For flat words $f_m = ()^m$, right-recursive concatenation gives $h(f_1) = 1$ and $h(f_m) = 1 + \max(1, h(f_{m-1})) = m$. For nested words $d_m = (^m)^m$, the $P \to LX$ and $X \to DR$ rules add two binary levels, so $h(d_1) = 1$ and $h(d_m) = 2 + h(d_{m-1}) = 2m - 1$. Both families thus require linearly many local rounds under this grammar, despite nesting depths of one and m. Local rounds measure grammatical derivation height, not bracket nesting depth.

A.2 REACHABILITY CORRECTIONS MEASURE STRAHLER DIMENSION

Throughout this subsection, the grammar is unit-free strict CNF. We express the reachability correction of Section 2.2 as a linear fixed point, then relate its dependency paths to derivation trees.

Linear correction. Fix a sound chart X and initialize the growing chart Y to X. Write $X_A[i,j] = 1$ when $A[i,j) \in X$, and zero otherwise. For a rule $A \to BC$, the correction may combine a child in Y with a sibling in X, giving the linear operator

$$(L_X Y)_A = \bigvee_{A \to BC} (Y_B X_C \vee X_B Y_C), \tag{10}$$

where juxtaposition is Boolean interval-matrix multiplication. Keeping X fixed, iterate $Y \leftarrow X \vee L_X Y$. Thus each binary inference uses at least one child from X, and one correction computes:

$$\mathcal{C}(X) = \mu Y.\,(X \vee L_X Y) = \bigvee_{d \geq 0} L_X^d X. \tag{11}$$

Here μ denotes the least fixed point: the smallest chart containing X and closed under L_X. As in Section 2.2, set $X^{(0)} = B(w)$ and $X^{(t+1)} = \mathcal{C}(X^{(t)})$.

Derivation spines and Strahler dimension. Draw a dependency edge from a parent item to one child whenever the other child belongs to X. Then $L_X^d X$ contains exactly the items with a length-d path into X. Every edge strictly shortens the span, so only finitely many path lengths occur. Such a path describes a derivation spine whose endpoint and off-spine children have derivations represented in X. The entire spine is resolved in one correction. Strahler dimension captures this structure: it increases when equally complex branches meet and stays unchanged when a simpler branch is attached. The Strahler dimension is $r(\text{leaf}) = 0$ and, at a binary node,

$$r(T) = \begin{cases} r(T_L) + 1, & r(T_L) = r(T_R), \\ \max(r(T_L), r(T_R)), & r(T_L) \neq r(T_R). \end{cases} \tag{12}$$

Proposition A.2 (Correction rounds and Strahler dimension)**.** *For a unit-free strict-CNF grammar and every $t \geq 0$, $X^{(t)}$ contains an item exactly when it has a derivation of Strahler dimension at most t. For $w \in L(G)$, the first correction round containing the root is* $\min_{T: S \Rightarrow^* w} r(T)$.

Proof of Proposition A.2. The lexical base is the $t = 0$ case. Suppose the claim holds for $t-1$. First, consider an item with a derivation of dimension at most t. If that dimension is at most $t-1$, the item already belongs to $X^{(t-1)}$. Otherwise its dimension is t, and at each binary node at most one child can have dimension t. Follow that child while it exists. At the final node both children have dimension at most $t-1$; take one further edge to either child. This gives a finite dependency path whose endpoint and all off-spine children belong to $X^{(t-1)}$ by the induction hypothesis. One correction therefore derives the root.

Conversely, take a dependency path into $X^{(t-1)}$. Its terminal tree and all sibling trees can be chosen with dimension at most $t-1$. Attach the sibling trees successively, starting at the path's endpoint. If the current spine tree has dimension at most $t-1$, the next attachment produces dimension at most t. If it has dimension t, a sibling of dimension at most $t-1$ leaves that dimension unchanged. The argument allows multiple derivations; the first correction deriving an item gives the minimum Strahler dimension among its trees. □

Round bound and implementation cost. A binary tree of dimension r has at least 2^r leaves. If the children have equal dimension $r-1$, their leaf counts add to at least 2^r; if unequal, the larger child already satisfies the induction bound. Thus a tree with ℓ lexical leaves has $r \leq \lfloor \log_2 \ell \rfloor$. Every item spans at most n terminals, so Proposition A.2 gives full chart saturation within $\lfloor \log_2 n \rfloor$ corrections for $n \geq 1$.

This counts semantic corrections, not transformer loops. Reducedness and unambiguity give the unique-path property needed by Lemma B.1; the implementation uses $O(\log n)$ loops per correction and $O(\log^2 n)$ loops overall (Theorem 3.2). The semantic bound holds with ambiguity, but unique-path contraction need not apply. Theorem 3.3 instead uses tree/context composition.

B Parsing constructions: details and proofs

We prove the three constructions of Section 3, separating algorithmic dependencies from their attention implementations. Each language-specific section gives both its public and blank realizations;

the shared initialization and normalized-execution lemmas are proved afterward. Asymptotic bounds concern $n \geq 2$; shorter inputs use fixed finite control.

B.1 Proof roadmap and interfaces

Shared tools. Section B.2 develops retrieval, aggregation, and causal layouts for all three constructions. The initialization-independent status contraction in Section B.3 evaluates the unique-path dependency graphs used by DCFL and UCFL parsing. For UCFLs, Proposition A.2 bounds the number of corrections, while Lemma B.1 bounds the rounds within each correction. Section B.4 supplies depth-compressed public matrix products for UCFLs and general CFLs.

Parsing constructions. The DCFL construction in Section B.5 merges interval summaries in reusable quadratic work banks to prove Theorem 3.1. The UCFL construction in Section B.6 combines unique-path contraction with shared arithmetic to evaluate reachability corrections, proving the public bound of Theorem 3.2 and its blank bound using the rank-seven identity. The general-CFL construction in Section B.7 composes trees and contexts through Boolean products to handle ambiguity without the unique-path assumption, proving Theorem 3.3. Each construction also gives its certificate recovery procedure.

Shared implementation and resource accounting. Section B.8 proves the shared initialization and execution guarantees used by the preceding constructions. Public realizations supply the static fields; blank realizations generate them internally. Section B.9 collects public-field generation costs, total attention work, and exponent conventions.

B.2 Addressing and causal layouts

The parsing constructions need address reads, missing-entry detection, and conditional status queries. The implementations below recover exact discrete results from normalized AHAT reads with constant width and $\mathcal{O}(\log n)$-bit precision. Blank constructions generate the required static fields internally, as specified in each application.

Unique retrieval requires a positive score gap between the target and all other readable records, allowing small address errors. Tied aggregation requires bit-identical scoring keys for the intended records, all scoring above the rest; approximate agreement alone is insufficient. Discrete updates keep Boolean and finite states exact while allowing bounded error in numerical directions.

B.2.1 Retrieval and discrete queries

Exact address retrieval. To retrieve a record with integer address a, encode its address as $\phi(a) = \mathrm{LN}(a, 1, -a, -1)$, where LN denotes layer normalization to zero mean and unit variance (Merrill & Sabharwal, 2025a). Use this encoding for both the query address and the stored keys. Each encoding has squared norm four: a matching address has dot-product score four, whereas every nonmatching address scores strictly less. Whenever matching records are present, AHAT averages their values. Concatenating a fixed number of these encodings handles tuple addresses and record types without increasing width with n.

Finite precision must preserve the separation between matching and nonmatching scores. For distinct integers $a, b \in [0, A]$, the score gap is

$$4 - \langle \phi(a), \phi(b) \rangle = \frac{4(a-b)^2}{\sqrt{(a^2+1)(b^2+1)}\left(\sqrt{(a^2+1)(b^2+1)} + ab + 1\right)} \geq \frac{2}{(A^2+1)^2}. \tag{13}$$

Rounding each coordinate to absolute error at most 2^{-p} changes a four-coordinate dot product by at most $16\,2^{-p}$ for $p \geq 1$. Thus $2^{-p} < 1/[64(A^2+1)^2]$ keeps matching scores above all nonmatching scores. Copies of the same address use identical rounded keys and retain exact ties. Polynomially bounded addresses need only $p = \mathcal{O}(\log n)$ bits; tuple addresses change only the precision constant.

Retrieval with missing entries. For every queried address, place a presence-zero default before its reader. There is at most one eligible real record with that address. Within the matching address group, give an eligible real record priority above the default and every ineligible record priority below

it. An inverse-polynomial priority smaller than the address gap preserves this order under rounding. The head selects either the real record or the default uniquely and returns its exact presence code and separately normalized payload directions. The concrete priority and comparison implementation is given in Section B.2.2; direction copies obey Lemma B.19.

Existence and zero/one/many queries. For a group of conditionally eligible records with distinct static identifiers, query the minimum and maximum eligible identifiers, using matching defaults. Two defaults mean zero records, one identical real identifier means one, and distinct extrema mean at least two. In the singleton case the selected record also supplies its address. Section B.2.2 implements these queries and identifier equality in constant stages through normalized priorities and guarded comparisons. Public initialization supplies the canonical labels, scales, and comparison fields; blank initialization generates them internally. DCFL contraction needs only these outcomes, not the numerical count.

Exact finite operations and arithmetic. Boolean and finite-state updates use balanced finite codes and the normalized controller of Lemma B.13. Public integer arithmetic uses separate bit records and constant-depth local circuits (Section B.4.2), including counts and split-weighted counts. Tied aggregation occurs inside their Boolean gates, where canonical group keys and equal-norm finite payloads satisfy Lemma B.19. Integer results are exact bit strings; the attention interface carries address directions and normalized finite codes.

Blank implementation interface. Under the decoder conditions of Lemma B.18, static fields are generated internally in $O(\log^2 n)$ loops, followed by a common execution start. Lemma B.19 preserves unique reads, exact ties, and finite decisions; tied reads require canonical keys and payload normalization that preserves key equality. The local circuits of Lemma B.20 compute exact output bits in constant depth once earlier input bits are ready, with potential-edge records charged to storage. DCFLs use extrema queries (Section B.5.3); UCFLs and general CFLs use local arithmetic (Sections B.6.3 and B.7.3).

B.2.2 Normalized extrema and equality

The queries above apply in both initialization modes once their static address directions and priority scales are available. The following implementation selects an eligible record by its identifier and compares the returned identifiers in constant stages.

Eligibility and extrema. A query first prioritizes the requested address, then eligibility, then the first coordinate of the identifier direction, with sign chosen for a minimum or maximum. Use a matching default with zero eligibility, eligible real rows with positive eligibility, and ineligible rows with negative eligibility. Eligibility is a balanced three-state one-hot, so its ideal norm is constant. All query, key, rank, and payload blocks have fixed ideal norms; inactive blocks have unit defaults. The head excludes unused fields.

Let $P = (n+2)^c$, with fixed c large enough to bound all labels and record identifiers, and let $\tau = 2^{-p}$ be the rounding scale of Section B.8.1. The rank coordinate is strictly increasing, with gap $\Omega(P^{-3})$ between adjacent identifiers. Write the score as $A(q,v) + \delta a(v) \pm \beta r(v)$, where A is the address-match score, a the eligibility code, and r the identifier's rank coordinate. Choose fixed-power scales $\delta \ll P^{-4}$, $\beta \ll \delta$, and total numerical error much smaller than βP^{-3}. Wrong groups lose to the matching default, which loses to any eligible matching row; the final priority makes the winner unique. Approximately copied query labels need not tie exactly.

In the public model these scales are supplied as rounded directions; in the blank model they are generated with a retained homogeneous anchor during initialization. Their magnitudes therefore survive normalization. For example, a stored direction for $(\delta, 1, -\delta, -1)$ has a first coordinate proportional to $\delta/\sqrt{1+\delta^2}$. A fixed query projection extracts this coordinate, while the key carries the finite eligibility code. Their dot product supplies the priority scale, up to bounded factors; no length-dependent weight or multiplication of two dynamic numerical fields is used. The same construction applies to β and the equality scale below.

Equality of retrieved identifiers. To compare the copied identifier directions x, y robustly, normalize $(x - y, \eta, -x + y, -\eta)$, with copy error $\ll \eta \ll P^{-2}$. Its positive marker is at least $1/2$ for equal labels and at most $1/4$ for distinct ones, with slack. Apply paired normalization to this whole vector and store it in a radius-two buffer. Its marker coordinate w is at least $1 - O(\tau)$ for equal labels and at most $1/2 + O(\tau)$ for distinct labels. The next sublayer applies the guarded sign construction to $4w - 3$; the constant comes from an immutable marker field. This gives an exact equality bit with a constant margin. Both projections are fixed and bias-free. Keeping the whole normalized vector in the buffer also permits the residual clearing argument of Lemma B.13, with no unnormalized marker register or affine bias.

B.2.3 Causal order and state reuse

A work bank is a set of token positions whose states are updated during execution. The DCFL construction updates fixed banks across loops; the UCFL and general-CFL constructions allocate later banks for successive stages.

Conditions for causal reuse. To reuse a work bank across loops, every read from another record must respect a fixed order: the source record precedes the record that reads it. A token retains its own state through residual updates and modifies it through pointwise operations. The execution schedule specifies when each source value is computed and keeps it available until its scheduled reads are complete. With these values available and all reads respecting the fixed order, subsequent loops can update the same token positions. Pointer compression also preserves the order: if position u points to $v < u$ and v points to $w < v$, replacing the pointer at u by w is still a read from an earlier position.

DCFL layout. For DCFLs, completed child-interval outputs precede the parent bank; within each bank, summaries with later input starts come first (Section B.5). The construction specifies the update schedule and the values retained for later reads.

B.3 Shared status contraction

This initialization-independent algorithm evaluates the dependency graph of one DCFL interval merge or one UCFL reachability correction. It resolves completed branches and compresses chains of single remaining dependencies, allowing information to propagate across many edges in one round.

Problem and required queries. The input is a finite simple acyclic dependency graph with at most one path between each ordered pair of vertices and a fixed set of base vertices. We seek all vertices that can reach a base. Bases are assigned true and their outgoing edges are ignored. For a nonbase vertex u, an edge $u \to v$ makes v a child of u; the value of u is the OR of its children's values. The graph and bases stay fixed throughout this computation.

Each vertex has status true, false, or unresolved. An unresolved vertex also has one of two modes: *waiting* queries its original children; *pointer* follows one descendant with the same final value. A waiting vertex needs three queries: whether any child is true; whether zero, one, or at least two children remain unresolved; and, in the one-child case, that child's address. These queries allow an implicit graph representation. Each vertex stores a status, a mode, and $O(1)$ pointers.

Contraction procedure. Initialize bases true, nonbase sinks false, and all other vertices unresolved in waiting mode. A vertex enters pointer mode when only one unresolved child remains and all others are false; it stays in this mode until it resolves. Each round performs the following steps in parallel across vertices. Each step reads a snapshot taken at its start and commits its updates synchronously; the two compressions in step 2 use successive snapshots.

1. Resolve waiting vertices. A true child makes the vertex true; if all children are false, make it false. Otherwise, point to the sole unresolved child when there is exactly one, or keep waiting when there are at least two. Existing pointer vertices are unchanged in this step.
2. Compress pointers twice in consecutive synchronous stages. When the target v is an unresolved pointer vertex, redirect $u \to v$ to its target $p(v)$; otherwise retain $u \to v$.
3. Resolve every pointer vertex whose current target is true or false by copying that status. Resolved vertices keep their statuses.

Lemma B.1 (Status contraction on unique-path graphs)**.** *Let a fixed simple acyclic graph on V vertices have at most one path between each ordered pair of vertices, with designated true bases treated as sinks. Suppose the three child-status queries can be answered exactly for all waiting vertices in parallel within $C \geq 1$ stages per round, and each synchronous pointer step takes constant stages. The contraction procedure evaluates base reachability at all vertices in $\mathcal{O}(\log V)$ rounds and $\mathcal{O}(C \log V)$ stages. If all nonlocal reads in the queries and pointer updates access earlier records, the procedure is strictly causal.*

Proof. (i) Correctness. Let $b(v)$ indicate whether v reaches a base in the fixed graph. We maintain two invariants: every resolved status equals $b(v)$, and every retained pointer $u \to v$ satisfies $b(u) = b(v)$. Initialization assigns the correct values to bases and nonbase sinks. Resolving a waiting vertex applies its OR relation. If only one unresolved child remains and all others are false, the vertex and that child have the same final value, so entering pointer mode preserves the invariants. Composing pointers preserves this equality, and copying a resolved target assigns the correct status. These observations prove the invariants by induction over the updates.

(ii) Number of rounds. Fix an unresolved root. At the start of a round, follow unresolved original children at waiting vertices and the current target at pointer vertices, omitting resolved targets. The reachable dependency graph is a tree: distinct branches from the root cannot meet again without violating the original graph's unique-path property, and pointer compression only shortens existing paths. This tree is an analysis device. A skipped vertex may remain unresolved and keeps its physical record, but no longer belongs to this root's current dependency tree. The tree's vertex set can only shrink: resolved statuses never change, and pointers advance along existing paths without returning to waiting mode.

Let the tree have $v > 1$ vertices, of which L are leaves, U have one child, and B have at least two children. Then $B \leq L - 1$. All L leaves resolve during the round because the dependencies they still consult are already resolved.

If $L \geq v/8$, at least an eighth of the vertices leave this tree. Otherwise $B < L < v/8$ and $U = v - L - B$. Suppressing the one-child vertices leaves a tree on $B + L$ vertices. Each maximal chain of one-child vertices occupies one of its $B + L - 1$ edges or the initial stem above its root, so there are at most $B + L \leq 2L$ such chains. Every vertex on these chains either resolves or has an active pointer after step 1. Within a surviving chain, the two synchronous compressions extend each pointer from one edge to two, then four, stopping at the chain's end. Thus at most $\lceil k/4 \rceil$ of its k vertices remain in the root's dependency tree. Since $k - \lceil k/4 \rceil \geq (k-1)/2$, summing over at most $2L$ chains removes at least

$$\frac{U - 2L}{2} \geq \frac{v - 4L}{2} > \frac{v}{4}$$

vertices from this tree. If the root resolves earlier, its evaluation is complete. Thus every unfinished round reduces its reachable tree to at most $7v/8$ vertices. The initial tree has at most V vertices, and a singleton resolves in the next round, giving $\mathcal{O}(\log V)$ rounds from any root. All vertices therefore resolve, and part (i) guarantees that their statuses equal the required reachability values.

(iii) Parallel evaluation and cost. The updates depend only on a vertex and its descendants. They therefore run on shared vertices for all roots simultaneously, without duplicating the per-root trees in storage. Pointer composition preserves the earlier-record order, establishing the causal claim. Each round uses $\mathcal{O}(C)$ stages, giving the stated total of $\mathcal{O}(C \log V)$. □

The applications supply these queries through DCFL helper records (Section B.5) or UCFL matrix products (Section B.6.2). Constant-stage implementations make $C = O(1)$, yielding $O(\log V)$ execution stages for one graph evaluation.

B.4 Shared matrix-product implementation

This public module implements the products used by UCFL corrections and general-CFL context composition. Integer operands and results are represented by exact bits in separate records; numerical directions serve only addressing and guarded aggregation. UCFL products return exact counts and split-weighted counts, while general CFLs test the final product for positivity.

Public initialization supplies circuit labels, coefficient bits, and canonical address fields. The blank applications instead decode their fields internally (Sections B.6.3 and B.7.3). Both use the normalized Boolean-gate execution proved in Section B.8. The rank-seven UCFL application returns exact counts without scale recovery; the general-CFL application retains a positive output scale.

Proposition B.2 (Public matrix-product module)**.** *Fix an exact rational matrix-multiplication identity with exponent $\rho > 2$ and an integer $k \geq 1$. Consider two $v \times v$ Boolean factors, allowing one factor to be weighted by split indices in $\{0, \ldots, v-1\}$. Their exact operand bits and ready flags occupy earlier banks, with polynomially bounded, logspace-computable port labels; ready operands remain fixed. With public circuit and canonical address fields, a fixed-parameter, constant-width strictly causal AHAT module returns the exact integer product as bits and ready flags using $O(v^{\rho+1/k})$ records, $O(k)$ stages once all inputs are ready, and $O(\log(v+2))$-bit precision. Each allocated module evaluates once and retains its completed outputs.*

We prove Proposition B.2 in the next three subsections, covering the circuit, exact arithmetic, and depth compression. Set $N = 2^{\lceil \log_2 \max\{v,2\} \rceil} = \Theta(v)$. Final product entries are at most N^2.

B.4.1 Bilinear circuits and integer scaling

Bilinear identity. Express matrix multiplication as r products of a linear form in the left inputs and a linear form in the right inputs, followed by linear combinations of those products (Landsberg & Ottaviani, 2015). Fix such an exact rational identity for side m. Clearing its denominators gives fixed integer arrays U, V, W and an integer $D \geq 1$ such that

$$D(AB)_o = \sum_{a=1}^{r} W_{oa} \left(\sum_b U_{ab} A_b \right) \left(\sum_c V_{ac} B_c \right). \tag{14}$$

The indices b, c, o enumerate matrix entries. Write $\rho = \log_m r$. The output-index flattening has rank m^2, so $r \geq m^2$ (Landsberg & Ottaviani, 2015, Section 2). We use fixed identities with $\rho > 2$; these can approach $\omega_{\mathbb{Q}}$ arbitrarily closely, even if the infimum is two, by tensor powering and, if necessary, adding a zero term.

Recursive circuit and bit bounds. Set $q = \lceil \log_m \max\{v, 2\} \rceil$ and zero-pad to side m^q. Recursive subproducts at depth $q-1$ carry a factor D^{q-1}, and the outer identity contributes another D, so the circuit returns $D^q AB$. Here D scales the whole identity in (14). Expanding the recursion yields two input linear networks, one scalar-product layer, and an output linear network. Fixed integer coefficients use constant-size binary addition/subtraction circuits. The size recurrence $T_q = rT_{q-1} + O(m^{2(q-1)})$ gives $O(v^\rho)$ gates for $\rho > 2$; each linear network has $O(\log N)$ depth. Coefficient growth and expanded path multiplicity are polynomial in N. Thus a fixed $b \log N + O(1)$ signed bit width covers every intermediate, including compressed partial sums and the scaled output. All subsequent circuits retain enough bits to avoid overflow.

B.4.2 Exact arithmetic in the normalized model

Public Boolean-circuit interface. Lemma B.20 supplies a common execution interface for both initialization modes. A fixed-depth Boolean/majority circuit with S gate and port labels occupies $O(S^2)$ records, including potential edges, defaults, and comparison anchors. Given exact input bits and ready flags in earlier banks, it returns exact output bits and ready flags in constant stages. Ready inputs remain fixed, and each allocated circuit evaluates once. Canonical keys preserve aggregation ties; guarded thresholds recover exact bits from rounded means at $O(\log N)$ precision (Lemma B.19). Public initialization supplies these static fields through the logspace generator in that lemma, so only execution contributes to the loop count.

Arithmetic and output recovery. Fix DLOGTIME-uniform constant-depth Boolean/majority circuits for signed addition, iterated addition, multiplication, comparison, and integer division (Hesse et al., 2002). Signs are finite bits; signed operations reduce to magnitude operations and Boolean selection. A weighted input $sL[i,s]$ is formed by ANDing $L[i,s]$ with each public bit of s. The same circuit interface sums the fixed number of grammar channels and tests zero, one, or positivity. These operations consume exact input bits and return exact output bits.

For a raw output $x = D^q y$, supply the bits of the positive integer D^q and use a division circuit to return y. The quotient is exact because the bilinear identity guarantees divisibility. With $O(\log N)$ operand bits, these final circuits use $\operatorname{polylog} N$ records per output and constant depth. A general-CFL product may instead test $x > 0$, since $D^q > 0$.

B.4.3 Depth compression and causal execution

Fix the arithmetic circuit family first. Let its depth be $D_0 = O(1)$ and size at most $S = C_0 R^C$ for at most R operands of R bits, including ports and signed arithmetic. The exponent C does not depend on the compression parameter. Given $k \geq 1$, choose $K = 2(2C+1)k$, set $t = \max\{1, \lceil (\log_2 N)/K \rceil\}$, and divide each binary linear network into chunks of at most t levels. Let Π contain the paths from a target to the chunk's input frontier. Expansion gives

$$y = \sum_{\pi \in \Pi} \sigma_\pi x_{s(\pi)}, \qquad \sigma_\pi \in \{-1, 1\}, \qquad |\Pi| \leq E = 2^t = O(N^{1/K}). \tag{15}$$

Paths reaching the same source stay separate, preserving signs and multiplicities. Pad to E paths with zeros. Since $\log N = O_K(E)$, a fixed multiple $R = C_K E$ covers both the operand count and the required bit width. One local circuit evaluates the signed sum; another implements each scalar product. Constants and source-bit ports have public labels.

Place each local circuit after its sources and before its consumers, using the causal copies of Lemma B.20. All sources for a chunk come from its input frontier, so the chunk's targets execute in parallel once those inputs are ready. A fixed cycle visits port reads, gate updates, and output latching; ready outputs remain fixed. There are $O(K) = O(k)$ chunks for the fixed identity, followed by constant-depth output recovery. This gives $O(k)$ execution stages.

Resource bound. There are $O(v^\rho)$ outer gates. For a uniform rectangular layout, reserve E slots per gate and $O(S^2)$ records per slot; unused slots remain inactive. This overestimates the one local sum circuit and its source ports, and gives

$$O\big(v^\rho E^{2C+1} \operatorname{polylog} N + v^2 \operatorname{polylog} N\big) = O(v^{\rho+1/k}).$$

Indeed $(2C+1)/K = 1/(2k)$ leaves $1/(2k)$ exponent slack for logarithmic factors, and $\rho > 2$ covers the output and bit-port banks. These costs include local edges, defaults, comparison anchors, and scale recovery. The complete product therefore has

$$M_{\mathrm{prod}}(v) = O(v^{\rho+1/k}), \qquad L_{\mathrm{prod}}(v) = O(k). \tag{16}$$

This completes the proof of Proposition B.2.

B.5 Deterministic parsing by reusable interval banks

We adapt the interval-halving and unique-path framework of Chytil et al. (1991, Theorem 2), which gives the classical quadratic-processor DCFL bound, to DPDA call summaries. Each merge of two neighboring intervals becomes a dependency graph evaluated by the shared solver in Section B.3. We specify the summary recurrence, graph construction, and reusable causal layout, retaining summaries for later merges and witness replay. We first establish the summary and graph properties independently of initialization, then give the public realization. Section B.5.3 generates its layout fields internally to obtain the blank-padding construction.

B.5.1 Call summaries and the merge graph

Summaries and their recurrence. To combine computations without replaying their internal transitions, we summarize individual stack calls of a fixed DPDA for the language. A call starts at a surface configuration $c = (q, i, A)$, specifying the control state, input position, and stack-top symbol, and returns when the stack first drops below its starting height. Its terminator $\tau(c)$ is the surface configuration immediately before that return pop; it is undefined if the call never returns (Glück, 2013). A true summary (c, e) satisfies $\tau(c) = e$; a candidate pair merely asserts this equality and must be checked. The stack below the starting frame is untouched before the return, so for a fixed input the call's behavior depends only on c.

We normalize transitions so that returns consume no input and every nontrivial call consumes input before starting its children. These properties will make child spans smaller and determine the causal record order. The two transition forms are a nonconsuming pop and a consuming replacement of the top symbol. By Autebert et al. (1997, Proposition 5.4), epsilon transitions can be chosen to pop the top symbol. We also replace each consuming pop by a consuming rewrite to a fresh marker followed by its epsilon pop. Thus every consuming transition replaces the top by $B_1 \cdots B_r$, listed in pop order, with $1 \le r \le R$ for a fixed automaton constant R. With fixed bottom and endmarker conventions, acceptance is reduced to a designated final pop.

These two transition forms give the base and recursive cases for summaries. An immediate pop gives $\tau(c) = c$, since the terminator precedes the pop. Otherwise, a consuming transition to state q' starts successive subcalls for $B_1, \ldots, B_r$, beginning with $c_1 = (q', i+1, B_1)$. Write $d_t = \tau(c_t)$ for $1 \le t \le r$. Each return determines the next start, and d_r is the parent's terminator:

$$c_{t+1} = (\mathrm{retstate}(d_t), \mathrm{pos}(d_t), B_{t+1}), \quad 1 \le t < r; \qquad \tau(c) = d_r. \tag{17}$$

Here $\mathrm{retstate}(d_t)$ is the state after the pop at d_t. Because this pop consumes no input, the next subcall starts at the same input position $\mathrm{pos}(d_t)$. If any required subcall has no terminator, the parent summary is undefined.

This recurrence supplies the interval structure used in the merge below. For a summary (c, e) with endpoint positions i, j, the initial consuming transition places every child within $[i+1, j]$, so each child has span strictly less than $j - i$. Since the children run successively, their input intervals are adjacent with disjoint interiors; immediate returns may have zero span. We write $C([a, b])$ for all true summaries with both endpoints in $[a, b]$. Determinism gives at most one terminator per surface start, so this set has $\mathcal{O}(b - a + 1)$ records for the fixed automaton.

Merging through a dependency graph. We compute $C([a, b])$ by combining the completed sets $C([a, h])$ and $C([h, b])$, where h is the interval midpoint. These sets already contain every true summary confined to either half; the merge must recover those whose start lies before h and whose end lies after it. In such a call, the successive subcalls have disjoint interval interiors, so at most one also crosses h. All other subcalls have summaries in the completed sets. For a true crossing call, any crossing child is therefore the first subcall missing from the completed sets: its prefix and suffix are known. If no child crosses, all subcalls are known and the parent follows directly.

We use candidate pairs (c, e) as logical vertices. Considering all possible endpoints gives quadratically many candidates even though the completed output has only linearly many true summaries. Known summaries and summaries obtained entirely from known subcalls form the bases. The following lemma constructs the dependencies so that a candidate belongs to $C([a, b])$ exactly when it can reach a base.

Lemma B.3 (Unique-path merge graph)**.** *Let $X = C([a, h]) \cup C([h, b])$ for $h = \lfloor (a + b)/2 \rfloor$, and let $m = b - a \ge 1$ be the interval span. The construction below defines an acyclic dependency graph with $\mathcal{O}(m^2)$ vertices and edges and at most one path between any two vertices. Its vertices are candidate summaries (c, e), and those reaching a designated base, including the bases themselves, are exactly $C([a, b])$. These graph properties do not depend on initialization. With public layout fields, the normalized reads of Section B.2 generate the graph from the two linear summary banks in constant stages using $\mathcal{O}(m^2)$ records.*

Proof. (i) Graph generation and size. View X as a partial function: $X(c)$ returns the unique known terminator for start c, or reports its absence. Store it in a linear source bank indexed by surface start, and allocate candidate records for all pairs (c, e) with endpoints in $[a, b]$.

For each start c, first check $X(c)$. If present, mark $(c, X(c))$ as a base and give all vertices with start c no outgoing edges. For each remaining start, require an enabled consuming transition within the interval; otherwise generate no base or edge. When this transition exists, follow Eq. (17) through children whose summaries are in X. If every child is found, mark the resulting parent as a base. If a lookup first fails at child t, retain its start $f(c) = c_t$, the index t, and the known prefix summaries. This first missing start depends only on c and X, not on the candidate parent endpoint e.

For each start with a missing child and each possible terminator d, allocate an edge helper (c, d) to store a possible dependency. Treat d as the missing child's candidate terminator, check that its return pop is enabled, and perform the remaining at most $R - 1$ lookups in X. Require all endpoints to lie

in $[a, b]$ and follow the input order of Eq. (17). If every check and lookup succeeds, define $F_c(d)$ as the parent terminator obtained by assuming that the missing child returns at d, and emit

$$(c, F_c(d)) \longrightarrow (f(c), d). \tag{18}$$

Otherwise $F_c(d)$ is undefined and the helper is inactive. Annotate each edge with the transition, child index, and known prefix and suffix summaries. The child starts strictly after c and ends no later than its parent, so every edge decreases span and the graph is acyclic.

All queries use the unique presence-flag retrieval of Section B.2; bank IDs separate sources from work records. For a summary shared at the midpoint, only its right-half copy is eligible in merge lookups. There are $\mathcal{O}(m)$ starts and $\mathcal{O}(m)$ possible terminators. Each helper uses $\mathcal{O}(R)$ sequential lookups with local intermediate values, and all helpers run in parallel. Since R is fixed, generation takes constant stages and $\mathcal{O}(m^2)$ records. Base flags are copied to the corresponding candidate records by exact address.

(ii) Correctness of the merge. For every emitted edge, a true child summary combines with the validated prefix and suffix to give a true parent summary. All bases are true, so induction backward along a path to a base proves soundness. Conversely, a true summary contained in either child interval is already a base. For a true summary crossing h, at most one subcall strictly crosses h; all other subcalls have summaries in X. If a crossing subcall exists, it is the first missing from X, and its edge is emitted. Otherwise the summary is a direct base. Recursing on the crossing child terminates by decreasing span, proving completeness.

(iii) Unique-path property. Fix two vertices (c, e) and (c', e'). Since f does not depend on the endpoint, every path from (c, e) follows the same sequence of starts $c, f(c), f^2(c), \ldots$. Their input positions strictly increase, so c' can occur at only one depth. Working backward from e', applying the corresponding F to each child endpoint uniquely fixes its parent endpoint. Thus the starts are fixed forward and the endpoints backward, determining at most one path. No injectivity of F is needed. The argument applies to all candidates, whether true or false. □

B.5.2 Public DCFL construction

Proposition B.4 (DCFL parsing with public initialization)**.** *For every fixed normalized DPDA, a fixed-parameter, constant-width strictly causal AHAT recognizes its language under public initialization using $\mathcal{O}(n^2)$ records, $\mathcal{O}(\log n)$-bit precision, and $\mathcal{O}(\log^2 n)$ loops on length-n inputs. On accepted inputs, it retains a replayable accepting computation annotated to recover a derivation in an associated fixed CFG.*

Proof. (i) Causal evaluation of one merge. We implement the queries of Lemma B.1 using the edge helpers from Lemma B.3. Order records by decreasing input position of c, placing helpers (c, d) before candidate records (c, e) at each position. Since $f(c)$ starts strictly after c, each child record precedes its helper, which precedes the parent record. Each round refreshes active helpers from a common child-status snapshot before updating parents. Queries exclude inactive helpers. All defaults and comparison anchors precede their readers.

A parent groups helpers by its address and requested status. Existence of a true helper resolves it; the minimum and maximum unresolved-helper identifiers distinguish zero, one, or multiple unresolved children and supply the sole child's address. The normalized extrema queries of Section B.2.2 implement these operations from public fields. A parent has at most one helper for each child $(f(c), d)$, so counting helpers up to two is equivalent to counting distinct children. These queries take constant stages. Each contraction round uses a fixed sequence of query, decision, and pointer-update stages, implemented by constantly many shared-block loops. Pointer updates use the successive synchronous snapshots specified in Section B.3. Compressed pointers still target calls with larger input start positions, hence earlier records.

Candidates, helpers, defaults, and comparison anchors together occupy $\mathcal{O}(m^2)$ records. With constant-stage graph generation and status queries, Lemma B.1 completes a merge in $\mathcal{O}(\log m)$ loops, reusing the same working records. A linear output bank after these records retrieves the unique true terminator for each start c, or a default indicating absence, and supplies $C([a, b])$ to the next merge.

(ii) Recursive composition and global bounds. Zero-span summaries are immediate pops. A summary spanning one input step uses one consuming transition followed by at most R zero-span subcalls, so

these base cases take constant stages. Recursively merge neighboring intervals using Lemma B.3 and the causal evaluation above. Both child banks and their outputs precede the parent bank. The children run in parallel; the parent begins only after both outputs are complete, and those outputs remain fixed throughout the merge. Within each bank, use the helper and candidate order established above. Induction over the recursion gives the exact summary set for the full input, so the designated accepting summary decides membership.

Let $\ell = n + \mathcal{O}(1)$ count input steps including fixed endmarkers, and choose a power of two n_{pad} with $\ell \le n_{\mathrm{pad}} < 2\ell$. Positions beyond ℓ are inactive. At depth r, the 2^r intervals each have span $n_{\mathrm{pad}}/2^r$ and contain $n_{\mathrm{pad}}/2^r + 1$ positions. Summing their work banks gives the storage bound below. Equal-size intervals run concurrently, so depth accumulates along one recursion path:

$$M(n_{\mathrm{pad}}) = \sum_{r=0}^{\log_2 n_{\mathrm{pad}}} 2^r \, \mathcal{O}\big((n_{\mathrm{pad}}/2^r)^2\big) = \mathcal{O}(n_{\mathrm{pad}}^2),$$

$$D(n_{\mathrm{pad}}) = D(n_{\mathrm{pad}}/2) + \mathcal{O}(\log n_{\mathrm{pad}}) = \mathcal{O}(\log^2 n_{\mathrm{pad}}).$$

The linear output banks total $\mathcal{O}(n_{\mathrm{pad}} \log n_{\mathrm{pad}})$ records; the schedule below adds $\mathcal{O}(\log^2 n)$ records. Both fit within the quadratic bound. Since $n_{\mathrm{pad}} = \Theta(n)$, these bounds are $\mathcal{O}(n^2)$ storage and $\mathcal{O}(\log^2 n)$ loops.

Place a public schedule before all execution banks. For intervals of span 2^s, reserve a fixed automaton-dependent multiple of $s + 1$ contraction rounds, sufficient by Lemma B.1; resolved states remain unchanged. Each schedule row specifies the interval size, finite execution phase, and next tick. Records start with the same local tick, retrieve its row, and copy the next tick in synchrony. Matching size and phase tags enable the intended updates while other fields retain their values. Thus the schedule depends only on input length and the fixed automaton. Its $\mathcal{O}(\log^2 n)$ rows encode the control sequence, while network weights remain fixed and shared across loops. Each record uses constantly many fields, including snapshots, counters, and witness annotations. Addresses, helper identifiers, and ticks are polynomially bounded. Schedule reads copy normalized tick directions and exact finite phase codes; level comparisons use the guarded equality test in Section B.2.2. The controller and runtime guarantees of Lemmas B.13 and B.19 therefore give $\mathcal{O}(\log n)$-bit precision for the full execution.

(iii) Accepting certificates. When a vertex enters pointer mode, save its chosen original edge and child address once; subsequent pointer compression and resolution leave these witness fields unchanged. A waiting vertex that resolves through a true child saves that edge instead. Such a child is unique because all child candidates share the start $f(c)$, which has at most one true terminator. A base stores either a reference to a completed child output or the transition and known subcalls that justify it. Replay from the accepting summary follows the saved edges and expands their transition, prefix, and suffix subcalls, continuing to immediate-pop bases.

The fixed DPDA induces a state-pair grammar: $[q, A, q']$ denotes a computation starting in state q with top symbol A and first removing that frame in state q'. An epsilon pop from (q, A) to q' gives $[q, A, q'] \to \varepsilon$. A consuming replacement $(q, a, A) \to (q_0, B_1 \cdots B_r)$ gives, for all return states,

$$[q, A, q_r] \to a[q_0, B_1, q_1] \cdots [q_{r-1}, B_r, q_r].$$

The start rule selects the initial frame and accepting final state, with fixed endmarkers erased from the terminal word. Saved transitions and child-return states identify the productions, recovering a derivation in this fixed grammar. An accepting run has $n + \mathcal{O}(1)$ consuming moves, each introducing at most R frames; all other moves pop a frame. The expanded computation and annotated derivation therefore have $\mathcal{O}(n)$ steps and $\mathcal{O}(n \log n)$ bits with explicit positions. Once expanded, they can be checked in $\mathcal{O}(n)$ word operations by replaying transitions, input consumption, and stack changes. Witness and grammar annotations add only constantly many logarithmic-bit fields per record, preserving the width, precision, and storage bounds in part (ii). □

B.5.3 Blank initialization and completion of the DCFL theorem

We realize the interval construction of Proposition B.4 from identical blank padding. The summary recurrence, contraction procedure, and witness semantics remain the same. We generate the layout and schedule internally; execution uses the same unique extrema queries. The decoder below supplies the fields required by the shared extrema and equality queries of Section B.2.2.

Direct decoding of the DCFL interval layout. Here m denotes interval span, with $m+1$ endpoint positions. Pad each interval bank to exactly Cm^2 slots, where C is a fixed constant large enough for its record types and endpoint ranges. Dummy slots have inactive types. Keep the two child banks before the parent, and keep decreasing starts and helper-before- candidate order within each parent bank. If m is a power of two, the subtree occupies

$$F(1) = C, \qquad F(m) = 2F(m/2) + Cm^2 = C(2m^2 - m). \tag{19}$$

For $m > 1$, compare a zero-based position $j < F(m)$ with $F(m/2)$ and $2F(m/2)$. The first two ranges descend to the left or right child, subtracting the corresponding offset; the third range is the current interval's bank. At $m = 1$, decode the constant-size leaf bank directly. This procedure recovers the interval and local slot in $\mathcal{O}(\log n)$ word-arithmetic steps and $\mathcal{O}(1)$ logarithmic-bit registers. Rectangularly padded start, endpoint, and type ranges then decode the local tuple by quotient and remainder. Thus the layout has the constant-word, logarithmic-operation decoder required by Lemma B.18, within quadratic storage.

Queries and causal reads. The queries of Section B.2.2 use the internally generated helper identifiers, address fields, and priority scales. Reads retain the child–helper–parent order of Proposition B.4; pointer compression preserves this order, and output records read completed work banks. The schedule prefix and matching defaults precede their readers. Every query includes immutable role and nonzero default fields. These heads select unique records, so controlled-error dynamic keys suffice. Copied fields are normalized before use, and finite-alphabet payloads recover exact statuses through fixed readout. Conditionals, writes, and buffer selection use Lemma B.13.

Proposition B.5 (DCFL parsing from blank padding)**.** *For every fixed normalized DPDA, a fixed-parameter, constant-width strictly causal AHAT recognizes its language from identical blank padding using $\mathcal{O}(n^2)$ records, $\mathcal{O}(\log n)$-bit precision, and $\mathcal{O}(\log^2 n)$ loops on length-n inputs, including initialization. On accepted inputs, it retains a replayable accepting computation annotated to recover a derivation in an associated fixed CFG.*

Proof. (i) Internal initialization. Use the interval banks of Proposition B.4. The decoder above initializes each interval's Cm^2 slots, including helpers, defaults, phase buffers, witnesses, and linear outputs. By (19), each static field takes $O(\log n)$ constant-word operations to decode. The schedule prefix reserves a fixed automaton-dependent multiple of $s+1$ rounds for intervals of span 2^s. Its $O(\log^2 n)$ rows decode their level and finite subphase by subtracting these level blocks, using $O(\log n)$ word operations. Therefore Lemma B.18 generates both layout and schedule in parallel, preserving the child–helper–parent order and releasing all records synchronously.

(ii) Execution and precision. Graph generation performs at most the fixed automaton constant R successive known-summary lookups. Dynamic endpoints are copied from terminators; the position increment $i+1$ after a consuming transition is a static start field generated at initialization. Control states and stack symbols come from fixed finite tables. The extrema heads implement exactly the three child-status queries of Lemma B.1. A parent has at most one helper for each child $(f(c), d)$, so testing for one helper also tests for one child. Pointer composition preserves the earlier-record order.

Use fixed subphases for helper reads, parent decisions, two synchronous pointer compressions, and resolved-target copying. A constant number of fields stores snapshots, and each interval bank is reused across rounds. After the common initialization barrier, synchronized local projective counters advance the tick and retrieve the schedule row; the clock-error argument of Lemma B.16 applies over the $O(\log^2 n)$ execution. The phase and level enable the intended updates through Lemma B.13. The unique-selection and comparison gaps in Section B.2.2 are inverse-polynomial, so Lemma B.19 preserves the selected records and exact finite decisions at $O(\log n)$ precision.

(iii) Resources and certificates. The replacement queries take constant stages with constant-width records. Thus the execution bounds of Proposition B.4 remain $O(n^2)$ records and $O(\log^2 n)$ loops. All banks initialize in parallel within $O(\log^2 n)$ loops and the same quadratic storage, giving the claimed total bounds. The merge semantics and saved original edges are unchanged, so the accepting certificate and grammar annotations of Proposition B.4 are retained. □

Completion of Theorem 3.1. Normalize a fixed DPDA recognizing the language as above. Proposition B.5 then gives the resource bounds and accepting certificate required by Theorem 3.1.

B.6 Unambiguous parsing by shared arithmetic circuits

Unambiguity makes correction graphs amenable to fast status contraction; shared matrix products implement its queries while controlling storage. We first establish the common semantics and convergence, then give the public construction and its blank realization using the rank-seven identity.

B.6.1 Correction graphs and convergence

Fix a reduced unambiguous grammar in strict CNF. The graph, uniqueness, and convergence arguments in this subsection are independent of the attention operator and of public versus blank initialization. We specify the status queries before choosing an implementation.

One correction as a dependency graph. Fix a sound chart X. Following Section 2.2, form a simple directed graph on all candidate items in $\mathcal{I}_n$. For each rule $A \to BC$ and split $i < k < j$, include $A[i,j) \to B[i,k)$ when $C[k,j) \in X$, and $A[i,j) \to C[k,j)$ when $B[i,k) \in X$; repeated edges are counted once. Keep X fixed throughout the correction. Its output $R_X = \mathcal{C}(X)$ consists of X and all items that can reach X. Each edge attaches a known sibling derivation, so R_X is sound.

Structural consequences of unambiguity. Unambiguity restricts both successful rule choices and paths in the dependency graph. A successful binary join $(A \to BC, i, k, j)$ has $i < k < j$ and both $B[i,k)$ and $C[k,j)$ in X.

Lemma B.6 (Successful-join injectivity)**.** *For a reduced unambiguous grammar and any sound chart X, the map $(A \to BC, i, k, j) \mapsto (A, i, j)$ is injective on successful binary joins.*

Proof of Lemma B.6. Take two successful joins with the same parent item. Soundness supplies derivation trees for both pairs of children. Distinct rules or splits give distinct parent trees with the same terminal yield. Since the grammar is reduced, the parent nonterminal occurs in a fixed productive start context. Substituting the two parent trees gives two distinct start trees for one word, contrary to unambiguity. A fixed rule and split already determine the two child addresses, so the joins are identical. □

This lemma gives uniqueness of rule/split witnesses. To apply status contraction, we also need a unique-path property covering all candidates, including unrealizable ones. Set $B_X = X \cup D(X)$ and treat these bases as sinks. Since every item in $D(X)$ reaches X in one step, reachability to B_X in the modified graph still computes R_X.

Lemma B.7 (Unique paths in the correction graph)**.** *For a fixed sound chart of a reduced unambiguous strict-CNF grammar, the correction graph with bases $B_X = X \cup D(X)$ treated as sinks is acyclic and has at most one path between each ordered pair of vertices.*

Proof. Each edge strictly shortens the span, so the graph is acyclic. A path specifies a partial derivation with its final descendant left as a hole; all sibling subtrees derive their corresponding input substrings. Two distinct paths with the same root and descendant give different partial derivation trees, but the same terminal strings to the left and right of the hole. By reducedness, the hole nonterminal has a terminal derivation. Fill both holes with that same tree, whose yield need not match the current input substring. The resulting trees are distinct and have the same yield. Embedding them in a common productive start context contradicts unambiguity. The same argument excludes distinct rule/split choices for a single edge. Removing base out-edges preserves path uniqueness. □

Status queries and repeated corrections. By Lemma B.7, Lemma B.1 applies. Each edge has a unique rule, split, and orientation, so counting these choices counts distinct dependencies. Let $T_A[i,j]$ and $U_A[i,j]$ indicate true and unresolved items, respectively. For a waiting nonbase item, the dependency counts are

$$c^T_A[i,j] = \sum_{A \to BC} \sum_{i<k<j} \big(T_B[i,k]X_C[k,j] + X_B[i,k]T_C[k,j]\big),$$
$$c^U_A[i,j] = \sum_{A \to BC} \sum_{i<k<j} \big(U_B[i,k]X_C[k,j] + X_B[i,k]U_C[k,j]\big). \tag{20}$$

The two terms count left-child and right-child dependencies, respectively: the queried child has its current status, while its sibling must belong to the fixed chart X. The solver uses only $\mathbf{1}[c^T > 0]$, $\mathrm{cap}_2(c^U) = \min(c^U, 2)$, and the child address and split when $c^U = 1$.

Within a correction, X, the bases, and the edges remain fixed while statuses and pointers change. On completion, replace X by R_X, recompute the bases, and initialize fresh statuses and pointers. A false status therefore means failure in the current correction; the item may become true in a later one. With $\mathcal{O}(n^2)$ vertices, Lemma B.1 gives $\mathcal{O}(\log n)$ contraction rounds per correction. Starting from lexical items, Proposition A.2 gives $\mathcal{O}(\log n)$ corrections for the full chart. The constant-stage queries and pointer updates in Section B.6.2 therefore yield $\mathcal{O}(\log^2 n)$ loops overall.

B.6.2 Public UCFL construction

Proposition B.8 (UCFL parsing with public initialization)**.** *For every fixed reduced strict-CNF unambiguous grammar G and fixed $\varepsilon > 0$, a fixed-parameter, constant-width strictly causal AHAT computes the exact chart and retains replayable rule/split witnesses under public initialization using $O(n^{\omega_{\mathbb{Q}}+\varepsilon})$ records, $O(\log^2 n)$ loops, and $O(\log n)$-bit precision on length-n inputs.*

Proof. (i) Matrix products for status queries. For a rule $A \to BC$, a left-child channel uses $L = T_B$ or U_B and $R = X_C$ in Eq. (20); a right-child channel uses $L = X_B$ and $R = T_C$ or U_C. Fix one such rule, orientation, and status channel. Its count and split-weighted count are

$$c_{ij} = \sum_{i<s<j} L[i,s]R[s,j], \qquad d_{ij} = \sum_{i<s<j} sL[i,s]R[s,j]. \tag{21}$$

Summing c_{ij} over rules and orientations separately for each status gives the total true and unresolved dependency counts. The solver tests whether the true count is positive and caps the unresolved count at two. If the unresolved total is one, exactly one unresolved channel is nonzero: its d_{ij} gives the unique split, and its rule and orientation identify the child address. The fixed grammar gives constantly many channels.

Use full endpoint matrices, with entries outside $i < j$ set to zero. Their ordinary product computes c_{ij}; replacing the left factor by $L^{\mathrm{wt}}[i,s] = sL[i,s]$ gives d_{ij}. The bit circuits of Section B.4.2 form these weighted factors before the call. The zero entries exclude splits outside $i < s < j$. Fix a rational identity with exponent $\rho > 2$ and an integer compression parameter $k \geq 1$. The product module of Proposition B.2 uses $O(n^{\rho+1/k})$ records and $O(k)$ stages per product, including all local Boolean-circuit records. Channel sums and comparisons return exact decision bits.

When the unresolved total is one, convert the selected split bits into an address for the sole child. Set $N = 2^{\lceil \log_2(n+1) \rceil}$. For each of $O(n^2)$ items, split the endpoint bits into k equal-length chunks, padding with leading zeros. Each chunk reserves $O(N^{1/k})$ candidates with public bit strings and canonical address directions. Constant-depth Boolean comparisons mark the unique match once all endpoint bits are ready; a group-first, match-second head copies its direction. Chunk keys and comparison anchors use Lemmas B.19 and B.20. Chart and pointer banks use the same tuple of k chunks per endpoint, so these directions address the earlier snapshot directly.

Conversion takes constant depth for fixed k and $O(n^{2+1/k} \operatorname{polylog} n) = O(n^{\rho+1/k})$ records since $\rho > 2$. Channel selection and comparisons add $O(n^2 \operatorname{polylog} n)$ records. Thus status queries, including child addresses, preserve the product module's storage and depth bounds.

(ii) Causal stages and resource bounds. Unique input-symbol reads and the fixed lexical rules produce the initial chart bits and mark them ready. Each contraction round reads a fixed chart and a snapshot of statuses and pointers, and produces the next snapshot. Place its banks in dependency order: the old snapshot, product factors and circuits, status decisions, two successive pointer compressions, and final status reads. Each pointer stage reads a complete snapshot retained in the preceding bank, including the fields at its target address. Thus the two compressions use successive snapshots as required by Lemma B.1, and all nonlocal reads access earlier records. Addresses, circuit fields, and finite role tags are public. Arithmetic stages use the ready protocol of Section B.4.3. Each pointer stage waits for its source record. In pointer mode it also waits for the selected target's ready flag before latching; other modes pass through without a target read. Final status reads use the same ready checks. Every completed record is marked ready and remains fixed, so these local checks preserve the logical snapshots. Reserve $O(\log n)$ contraction rounds per correction, as guaranteed by Lemma B.1.

At the start of each correction, products with both factors from X compute $D(X)$ and initialize the bases $X \cup D(X)$ as true. Let $J[i,j] = \mathbf{1}[0 \le i < j \le n]$. For each rule $A \to BC$, $JX_C + X_BJ$ counts outgoing dependencies; summing over rules identifies zero-degree vertices. Nonbase sinks start false, and all other nonbase vertices start unresolved in waiting mode. These initialization queries use constantly many additional products. Each contraction round likewise makes constantly many product calls and pointer updates. By Proposition A.2, $O(\log n)$ corrections suffice, giving $O(\log^2 n)$ loops for fixed k. Allocating fresh banks for all $O(\log^2 n)$ calls multiplies the per-product storage bound by this factor. Including the $O(n^2)$ snapshot and pointer banks per round, the total is

$$O\big(n^{\rho+1/k}\log^2 n + n^2 \log^2 n\big).$$

The constantly many witness products in part (iii) fit this bound as well. Integer outputs are exact bit strings. The normalized gate and address interfaces preserve them and their subsequent reads at logarithmic precision.

Choose the fixed identity with $\rho < \omega_{\mathbb{Q}} + \varepsilon/3$ and then fixed k with $1/k < \varepsilon/3$. The remaining slack absorbs the logarithmic factors. All public, working, input, and witness records fit $O_{G,\varepsilon}(n^{\omega_{\mathbb{Q}}+\varepsilon})$.

(iii) Rule and split witnesses. Once the full chart X is fixed, run the same products for each rule with $L = X_B$ and $R = X_C$. The outputs in (21) are now the successful-split count and split-weighted count. Completeness and Lemma B.6 give every true nonlexical parent exactly one successful rule, with count one; its weighted count is the split. The rule determines the two child nonterminals, and the parent endpoints and split determine their addresses. The conversion in part (i) supplies their endpoint directions. A later $\mathcal{O}(n^2)$ output bank retains these fields. Lexical witnesses use the fixed terminal rules. The additional product calls fit the bounds in part (ii), and the stored witnesses give replayable trees because child spans strictly decrease. □

B.6.3 Blank UCFL construction using the rank-seven identity

We retain the stage layout and count interfaces of Proposition B.8. The local Boolean circuits of Lemma B.20 supply exact arithmetic with internally generated fields. This construction applies to every grammar in Theorem 3.2; rank seven specifies its multiplication identity.

Proposition B.9 (Blank-padding rank-seven construction)**.** *For every fixed reduced strict-CNF unambiguous grammar and fixed $\varepsilon > 0$, a fixed-parameter, constant-width strictly causal AHAT constructs the exact chart and rule/split witnesses from identical blank padding with $M(n) = O(n^{\log_2 7+\varepsilon})$, $L(n) = O(\log^2(n+2))$, and $O(\log(n+2))$-bit precision. This includes initialization.*

Proof. (i) Rank-seven layout and static decoding. Strassen's identity (Strassen, 1970) has $m = 2$, $r = 7$, and $D = 1$. The identity is

$$\begin{aligned}
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), & M_2 &= (A_{21} + A_{22})B_{11},\\
M_3 &= A_{11}(B_{12} - B_{22}), & M_4 &= A_{22}(B_{21} - B_{11}),\\
M_5 &= (A_{11} + A_{12})B_{22}, & M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}),\\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}), & &\\
C_{11} &= M_1 + M_4 - M_5 + M_7, & C_{12} &= M_3 + M_5,\\
C_{21} &= M_2 + M_4, & C_{22} &= M_1 - M_2 + M_3 + M_6.
\end{aligned} \tag{22}$$

Expansion verifies each output entry.

Put $N = 2^{\lceil \log_2(n+1)\rceil}$, $d = \log_2 N$, $\rho = \log_2 7$, $g = \lceil d/(2k)\rceil$, and $E = 4^g = O(N^{1/k})$, for a fixed k chosen below. The rank-seven input forms have at most two terms and output forms at most four. Order input levels upward, scalar products next, and output levels downward. Reserve E relay slots and a target per scalar linear form, retaining inactive intermediate levels. Compressed segments have at most g levels. There are $O(N^{\rho+1/k})$ physical records and $O(k)$ active arithmetic segments. Expanded paths retain their signs and multiplicity; each source precedes its target.

A label decoder scans block lengths $2 \cdot 7^\ell (N/2^\ell)^2 (E+1)$ for the two input networks and the corresponding single-network lengths for the output, in reverse level order. Quotients and remainders identify rank path, row, column, and relay slot. At most g base-seven rank digits and binary or base-four path choices traverse the fixed coefficient lists to locate a relay's source. Unused path slots carry zero. Powers and block sizes are accumulated in one pass using constantly many words.

The full record decoder first identifies the correction, contraction round, and bank, then the outer module and its local gate, potential edge, or bit port. Conversion banks similarly decode an item, chunk, candidate, and bit position. Rectangular padding gives fixed-stride quotient/remainder formulas; local uniform predicates and external ports use the outer source decoder above. These calls have constant nesting depth and use $O(\log n)$ word operations in total. All decoded labels and coefficients therefore meet the static-field conditions of Lemma B.18.

(ii) Exact arithmetic and address conversion. The input-position reads of Lemma B.14 and the fixed lexical rules generate the initial chart bits in an earlier bank. Padded endpoints and nonincreasing spans have membership zero. A weighted factor $sL[i, s]$ is generated by ANDing $L[i, s]$ with each statically decoded bit of s. Thus both ordinary and weighted products receive exact input bits from internally generated fields.

The weighted input entries have magnitude at most N, and the other factor is Boolean. The input networks have fan-in at most two per level, so their linear forms have magnitude at most N^2 and N, respectively. Scalar products are bounded by N^3; at most $4^d = N^2$ output paths give an absolute-sum bound N^5, also covering compressed partial sums. A fixed $c_G(d + 1)$ signed bit width therefore suffices for these values and the sums over grammar channels. For each compressed signed sum or scalar product, uniform integer arithmetic (Hesse et al., 2002) supplies a constant-depth Boolean/majority circuit. Choose $R = C_{G,k}E$ large enough for both the operand count and this bit width. Such a fixed constant exists since $d = O_k(E)$. Zero-padding operand lists and sign-extending values gives at most R operands of R bits; retain sufficient output bits to avoid overflow. Fix the circuit-size exponent C and use $S = C_0R^C$ identifiers, including bit ports. The same family sums channel counts, tests zero/one/positivity, and selects a sole unresolved channel.

The decoders above and the uniform local ports meet Lemma B.20, which supplies internal edge generation, neutral nonedges, exact ready flags, and canonical tied aggregation. The reserved relay factor E and the local S^2 edge bound give $O(N^{\rho+(2C+1)/k})$ product records before stage-copy and logarithmic bit factors. Since $D = 1$, completed products give the exact count and split-weighted count as bits, without denominator normalization.

During contraction, a waiting item creates a new pointer only when $c^T = 0$ and $c^U = 1$. Its unique unresolved channel then supplies the endpoint $d_{ij} \in \{0, \dots, N - 1\}$. A true child instead resolves the item immediately. Existing unresolved pointer-mode records retain their pointers for compression; unused new-pointer slots remain inactive. Pad the selected endpoint to $4t$ bits, where $t = \lceil d/4 \rceil$, and represent it by four chunk directions. For each chunk, reserve $2^t = O(N^{1/4})$ candidates with internally initialized canonical directions. Each candidate has t comparison records that read and compare the endpoint bits in parallel. An AND over these exact comparison bits marks the unique match once all inputs are ready. As in Lemma B.20, finite-valued existence queries check readiness before enabling a single value aggregation into retained scratch. Canonical group keys and finite payloads satisfy Lemma B.19. A subsequent head scores candidate-type, item, chunk, and bank tags first, then the match flag within that group. An internally generated inverse-polynomial scale makes the flag's score contribution smaller than the group-matching gap; sufficiently large logarithmic precision preserves both priorities. The head retrieves the matching candidate's direction.

Chart-record keys use the same four-chunk encoding for each endpoint. Together with the nonterminal and snapshot tags, these form a fixed-size address tuple; a pointer read queries the designated earlier snapshot. This tuple directly addresses chart records at constant width. The candidate and bit-comparison records, including readiness tests, cost $O_G(N^2\, 2^t \operatorname{polylog} N) = O_G(N^{9/4} \operatorname{polylog} N)$ over the conversion banks and take constant execution depth per conversion. Subsequent pointer updates copy the chunk directions through unique reads.

(iii) Causal execution and witnesses. Allocate fresh banks in dependency order for each correction and contraction round: the old snapshot, arithmetic sources, local circuit copies, status decisions, chunk-comparison and candidate records, two pointer-update banks, and the new statuses. Pointer updates use the public construction's mode-dependent ready checks. Each reads completed values from one preceding snapshot, giving the two synchronous compressions required by Lemma B.1. Internal gate and edge copies follow the causal layout of Lemma B.20. Every nonlocal read is therefore to an earlier record.

At each correction's start, Boolean arithmetic computes $D(X)$ and the outdegree products of Section B.6.2, using internally decoded bits of J. Bases take priority, followed by nonbase sinks and

waiting states as in Lemma B.1. Exact ready flags permit a record to latch only after its genuine inputs are ready. A fixed finite phase cycle visits the module types and retains latched outputs. For fixed k, each product, status query, and pointer step has constant depth. The resulting dependency DAG has depth $O(\log^2 N)$: $O(\log N)$ contraction rounds per correction by Lemma B.1, and $O(\log N)$ corrections by Proposition A.2. All banks are initialized in parallel and share the internal release of Lemma B.18.

After the chart is complete, the same products return successful-split counts and weighted counts for each rule. Completeness and Lemma B.6 give a unique successful rule and split for every true nonlexical item. The conversion above supplies its child addresses in a later witness bank; lexical witnesses use the fixed terminal rules. Child spans strictly decrease, so the stored choices give replayable trees.

(iv) Resource and precision bounds. Including the $O(\log^2 N)$ parsing stages, witnesses, and bit records, the total storage is

$$O\Big(N^{\rho+(2C+1)/k}\operatorname{polylog} N + N^{9/4}\operatorname{polylog} N\Big). \tag{23}$$

Choose the arithmetic family first, then k so $(2C+1)/k < \varepsilon/3$. Since $9/4 < \rho$, the remaining exponent slack absorbs all polylogarithmic factors, giving $O(n^{\rho+\varepsilon})$ records. The constant-word decoders and common release give $O(\log^2(n+2))$ initialization loops by Lemma B.18; execution has the same bound.

Arithmetic outputs, comparisons, and ready flags remain exact bits. Static candidate directions and aggregation keys use the canonical construction of Lemma B.17; dynamic pointer directions are used only for unique reads and are excluded from tied aggregation. All labels are polynomially bounded, and address, priority, and threshold gaps are inverse polynomial. Taking the larger precision constant from Lemmas B.18 and B.19 gives one $O(\log(n+2))$ bound for initialization and execution, with constant width and fixed parameters. □

Completion of Theorem 3.2. Propositions B.8 and B.9 give its Public and Blank parts, respectively.

B.7 General parsing by tree–context composition

General correction graphs can have merging paths and multiple successful joins. We instead compose complete trees and one-hole contexts using the classical doubling algorithm of Brent & Goldschlager (1983). We first prove its logarithmic convergence, independently of initialization, then give public and blank AHAT implementations. Both reduce composition to Boolean products on $\Theta_G(n^2)$ items, yielding the doubled matrix-multiplication exponent. Their arithmetic needs only the positivity of completed products (Section B.4).

B.7.1 Tree/context doubling

Let $v = |\mathcal{I}_n| = \mathcal{O}_G(n^2)$. A one-hole context from $u = A[i,j)$ to $z = B[p,q)$ is a partial parse rooted at A that produces $w_i \cdots w_{p-1}$, the unexpanded occurrence B, and $w_q \cdots w_{j-1}$. Its hole lies within its outer span. The hole need not itself have a complete derivation. The Boolean vector P_t and matrix Q_t are indexed by items: $P_t[u] = 1$ records an established complete derivation of u, and $Q_t[u,z] = 1$ an established context from outer item u to hole item z. Initialize P_0 to the lexical chart and $Q_0 = I_v$, the identity contexts consisting only of their hole.

For a Boolean chart P, let $E(P)$ be the parent-to-child adjacency matrix of the correction graph in Section 2.2: each edge represents a one-production context whose sibling item is in P. Let $D(P)$ be the indicator vector of the binary consequences of P. Write $\vee$ for entrywise OR and $\odot$ for Boolean matrix or matrix–vector multiplication. One phase consists of

$$\begin{aligned} U_t &= (I_v \vee E(P_t)) \odot Q_t, \quad & Q_{t+1} &= U_t \odot U_t, \\ V_t &= P_t \vee D(P_t), & P_{t+1} &= Q_{t+1} \odot V_t. \end{aligned} \tag{24}$$

The first operation either retains a context or wraps it in one production with a known sibling. The second composes contexts. The final operation fills a context with a known tree or an immediate binary consequence.

Lemma B.10 (Soundness and logarithmic convergence)**.** *Fix a strict-CNF grammar and an input of length $n \geq 1$. Starting from the lexical chart P_0 and $Q_0 = I_v$, (24) preserves soundness and all established bits, and $P_{\lceil \log_2 n \rceil}$ is the exact chart.*

Proof. Each operation composes valid partial or complete derivations, preserving soundness. Identity contexts remain present, so Q_{t+1} contains Q_t and P_{t+1} contains P_t.

Count nonterminal occurrences as nodes, treating lexical occurrences and the distinguished hole as leaves. A complete tree on ℓ input symbols has $2\ell - 1$ nodes; a context has $2[(j - i) - (q - p)] + 1$ nodes. We prove simultaneously that Q_t marks the outer/hole pair of every valid context of size at most 2^t, and P_t marks the root item of every valid complete tree of size less than 2^{t+1}. Both statements hold at $t = 0$.

For a nontrivial context T of size $s \leq 2^{t+1}$, follow the path to its hole until finding a node c whose subtree has more than $s/2$ nodes but whose child on that path has at most $s/2$. Such a node exists because the hole has size one. Let s_c be the size at c. Replacing that subtree by a hole leaves an outer context of size $s - s_c + 1 \leq (s + 1)/2 \leq 2^t$, since $s_c > s/2$ and s is odd. The child on the hole path gives an inner context of size at most $\lfloor s/2 \rfloor \leq 2^t$. The other child is a complete sibling tree of size less than 2^{t+1}. By induction Q_t marks both context pairs and P_t marks the sibling item. Wrapping the inner context places its pair in U_t; the outer context is also retained in U_t. Their composition marks the pair for T in Q_{t+1}.

For a nonlexical complete tree T of size $s < 2^{t+2}$, descend through a child of size greater than $s/2$ while one exists, keeping s fixed. At the stopping node c, its subtree size s_c still exceeds $s/2$, but both children have size at most $s/2 < 2^{t+1}$. Thus P_t marks both child items and V_t marks the root item at c. Replacing this subtree by a hole leaves an outer context of size $s - s_c + 1 \leq (s + 1)/2 \leq 2^{t+1}$. Its pair is therefore marked in Q_{t+1} by the context claim just established. Filling it marks the root of T in P_{t+1}. Lexical trees are retained. It follows that $P_{\lceil \log_2 n \rceil}$ is the entire chart. □

B.7.2 Public implementation and resource bounds

Proposition B.11 (Causal AHAT implementation of tree/context doubling)**.** *For every fixed strict-CNF grammar G, and every fixed $\varepsilon > 0$, there is a constant-width, logarithmic-precision strictly causal AHAT in the public-table model that computes the exact chart and retains one parse witness per realizable item with*

$$M(n) = \mathcal{O}_{G,\varepsilon}(n^{2\omega_{\mathbb{Q}}+\varepsilon}), \qquad L(n) = \mathcal{O}_{G,\varepsilon}(\log n).$$

Proof. (i) Boolean products and causal banks. Fix a rational rank-r identity on side m with $\rho = \log_m r > 2$, and a compression parameter k. The Boolean factors have been computed in earlier banks and remain fixed throughout the call. Proposition B.2 computes their integer product as exact bits using $\mathcal{O}(v^{\rho+1/k})$ records and constant depth for fixed identity and k. Entries lie in $[0, v]$; a local positivity circuit returns the Boolean product. This module requires neither unique paths nor span-ordered matrix indices. Matrix–vector products use one nonzero input column.

Input-symbol reads supply lexical bits, and public item indices supply identity bits; both are marked ready once set. Each phase of (24) uses constantly many products. Each entry of $E(P_t)$ inspects constantly many sibling bits: the parent and designated child determine the split, leaving only the fixed grammar rules to check. Computing $D(P_t)$ uses one Boolean product of endpoint matrices of side $n + 1$ per binary rule. Allocate fresh banks for every phase and suboperation, placing factors before their product modules and outputs. All reads are strictly earlier. Ready flags activate successive banks under the shared block; completed banks retain their values.

(ii) Resources and precision. By Lemma B.10, there are $\mathcal{O}_G(\log n)$ products. For the fixed grammar, identity, and k, substituting $v = \Theta_G(n^2)$ gives total storage and loop depth

$$\mathcal{O}(n^{2\rho+2/k} \log n + n^4 \log n), \qquad \mathcal{O}(\log n).$$

The first storage term includes all local bit circuits across phases; the second counts the $O(v^2 \log n)$ relation records. The context relation can have $\Theta(n^4)$ true entries, as for $S \to SS \mid a$ on a^n. Choose ρ close enough to $\omega_{\mathbb{Q}}$ and then k large enough that $2\rho + 2/k < 2\omega_{\mathbb{Q}} + \varepsilon/2$. The remaining polylogarithmic factors are absorbed by $n^{\varepsilon/2}$, also when $\omega_{\mathbb{Q}} = 2$. Each completed product is

thresholded before any subsequent operation, including within the same phase, so arithmetic operates on Boolean relations rather than accumulated parse counts. Intermediate integers remain polynomially bounded and are stored as bits; Section B.4 supplies logarithmic-precision gate execution. The public-field generator in Section B.9 enumerates the banks and their circuit fields in logarithmic space.

(iii) Witnesses. Place $\mathcal{O}_G(n^3)$ rule/split candidate records after the completed chart, followed by a bank of $\mathcal{O}_G(n^2)$ parent witness records. Each candidate reads its two child bits and marks whether the join succeeds. A parent witness record then reads only earlier candidates and a matching default. Within its parent group, successful candidates outrank the default, which outranks failed candidates; among successful candidates, the normalized minimum head of Section B.2.2 selects the least public rule/split index. Its rank direction lies below the parent and eligibility priorities, preserving an inverse-polynomial retrieval gap. The output record stores the chosen production and child addresses; lexical items retain their terminal rule. Child spans strictly decrease, so the selected pointers give valid parse trees even when other parses exist. Both banks fit the bounds and add constant depth. □

B.7.3 Blank implementation and completion of the theorem

To implement the doubling recurrence from blank padding, we give each fixed rational multiplication identity an internally decodable layout and implement its arithmetic with the local Boolean circuits of Section B.8. The following lemma supplies the product modules; we then assemble the parsing phases and witness bank.

Lemma B.12 (Internally initialized Boolean products)**.** *Fix a rational rank-r matrix-multiplication identity of side m with $\rho = \log_m r > 2$, and fix $\delta > 0$. Consider polynomially many product modules of side v. Suppose their bank and external bit-port addresses, including ready flags, are decoded by a fixed program using $O(\log(v+2))$ word operations on constantly many polynomially bounded words, with constant call depth. A strictly causal AHAT with fixed width and parameters can initialize these modules from blank padding in parallel, using $O(v^{\rho+\delta})$ records per module, $O(\log^2(v+2))$ loops, and $O(\log(v+2))$ precision. Once initialized and supplied with fixed exact operand bits and ready flags in earlier banks, each module returns the exact Boolean product and ready flags in constant loop depth.*

Proof. (i) Integer coefficients and exact positivity. Write the fixed identity as $U, V \in \mathbb{Q}^{r\times m^2}$ and $W \in \mathbb{Q}^{m^2\times r}$. Choose a fixed positive common denominator D and replace these matrices by $U' = DU$, $V' = DV$, and $W' = DW$. Pad the input matrices with zeros to side $N = m^d$, where $d = \lceil \log_m \max\{v, 2\} \rceil$. Each of the two input linear networks acquires a factor D^d and the output network another D^d. Consequently its exact integer output is

$$\widetilde{C} = D^{3d}AB, \qquad (A \odot B)_{ij} = \mathbf{1}[\widetilde{C}_{ij} > 0]. \tag{25}$$

The exponent $3d$ comes from scaling all three coefficient arrays at each level; the D^q convention in (14) scales the whole identity once per level. No division by D^{3d} is required. Signed intermediates and path multiplicities are retained until the final test. If H bounds the fixed integer coefficients and $s = \max\{m^2, r\}$, every intermediate has magnitude at most $(s \max\{H, 1\})^{O(d)}$. Thus $b(d+1)$ bits suffice, including a sign bit, for a fixed constant b.

(ii) Compressed layout and address decoding. Fix an integer $a \geq \log_m s$ and an integer compression parameter $k \geq 1$. Set $g = \max\{1, \lceil d/(ak) \rceil\}$ and $E = s^g = O(N^{1/k})$. Compress at most g consecutive linear levels into one weighted sum of at most E paths; there are $O(ak)$ active segments. Retain inactive levels and reserve E relay slots and one target per scalar linear gate. Input levels increase, scalar products follow, and output levels decrease. Since

$$\sum_{\ell=0}^{d} r^\ell (N/m^\ell)^2 = O(N^\rho),$$

this layout has $O(N^{\rho+1/k})$ records before Boolean compilation. Each nonzero relay reads a target in an earlier active segment.

A decoder scans block lengths $2r^\ell(N/m^\ell)^2(E+1)$ for the input networks and the corresponding single-network lengths for the output. Quotients and remainders identify rank path, row, column, and relay slot. At most g base-s choices and base-r rank digits traverse the fixed coefficient lists.

Coordinates are updated in base m, and coefficients are multiplied along the path. Zero coefficients deactivate a relay. These finite lists are part of the fixed program, not length-dependent padding contents. Powers and block sizes are accumulated in a pass. The decoder uses $O(\log N)$ word operations, constantly many polynomially bounded words, and constant call depth.

(iii) Local circuits, output, and resources. Fix DLOGTIME-uniform Boolean/majority circuits for integer multiplication, iterated addition, and positivity (Hesse et al., 2002, Section 2.2). Their depth D_0 and polynomial-size exponent C are fixed before choosing k. Each weighted sum has at most E operands, and part (i) bounds the required bit widths by $b(d+1)$. Since $d \le akg \le akE$, a fixed constant c depending on the identity and k makes $R = cE \ge \max\{E, b(d+1)\}$. Zero/sign padding therefore embeds the computation into one with at most R operands of R bits. The circuit size is $S = C_0 R^C$, including ports and signed arithmetic; scalar products and final positivity tests obey the same bound. Retain all bits required by part (i) to avoid overflow.

Apply Lemma B.20 to each local circuit. Part (ii) locates an operand's source scalar gate in an earlier active segment; the source label and bit index identify its output-bit port and ready flag. At the matrix-input boundary, use the assumed external-port decoder. A bounded bit scan supplies the signed path coefficient, with zero/sign extension for padded ports. Together with the local uniform predicates, these decoders use $O(\log N)$ word operations and constant call depth. The shared lemma supplies internal edges, neutral nonedges, exact ready propagation, and guarded value aggregation. Each allocated circuit evaluates once after its genuine inputs are ready. The final positivity test returns exact Boolean bits and ready flags as the product's output.

The outer layout has $O(N^\rho E)$ slots. Reserving $O(S^2) = O(E^{2C})$ records per slot suffices for the local circuits. Including logarithmic factors, total storage is

$$O(N^\rho E\, S^2 \operatorname{polylog} N) = O(N^{\rho+(2C+1)/k} \operatorname{polylog} N).$$

Outer bank and module indices have fixed-stride quotient/remainder descriptions. With the arithmetic family and C fixed, choose k so $(2C+1)/k < \delta/2$ and absorb the polylogarithm into the remaining slack. Since $N = \Theta(v)$, this gives $O(v^{\rho+\delta})$ records per module. There are constantly many compressed segments for fixed k, giving constant runtime depth once operands are ready. Coefficients and addresses are polynomially bounded. Lemma B.18 initializes all banks in parallel in $O(\log^2(v+2))$ loops; Lemmas B.19 and B.20 give exact outputs at logarithmic precision. □

Proof of Theorem 3.3. The public case is Proposition B.11. For the blank case, pad the legal item set to the rectangular grid $\mathcal{N} \times \{0, \dots, n\}^2$, marking $i \ge j$ invalid. Here v denotes the padded grid size, still $\Theta_G(n^2)$, with constant-word quotient/remainder decoding. The input-position reads of Lemma B.14 and fixed terminal rules generate the lexical chart P_0. Decoded valid-item labels generate the diagonal bits of Q_0; invalid rows and columns are zero. Mark these initial bits ready when complete. Matrix coordinates and bit indices also determine the operand ports of each later module. Use Lemma B.12 for every product in (24); matrix–vector products use one nonzero column. The endpoint products for $D(P)$ can be zero-padded to the same side without changing the exponent. Each entry of $E(P)$ reads the constantly many sibling bits identified in the public construction.

By Lemma B.10, $O(\log n)$ phases complete the chart. Allocate fresh banks in dependency order for every phase: old relations, product input ports, local edge/gate copies, product outputs, and new relations. Each module evaluates once after its inputs are ready, and all reads are strictly earlier. Constant-depth product modules give $O(\log n)$ execution loops for the fixed identity and parameters. The static fields of every phase bank are initialized once, in parallel, in $O(\log^2 n)$ loops.

Witness selection follows Proposition B.11, with internally decoded candidate indices and child addresses. The eligibility-prioritized minimum head of Section B.2.2 implements the selection in the later parent bank. This uses $O_G(n^3 \operatorname{polylog} n)$ records and constant additional execution depth; replayability follows from the public proof.

Applying Lemma B.12 to the $O(\log n)$ phases gives total storage

$$O_{G,m,r,\delta}\left(n^{2\rho+2\delta} \log n + n^4 \operatorname{polylog} n + n^3 \operatorname{polylog} n\right). \tag{26}$$

The terms account for product modules, relation banks, and witnesses. Choose a fixed rational identity and $\delta > 0$ such that $2(\rho - \omega_{\mathbb{Q}}) + 2\delta < \varepsilon$. Since $\omega_{\mathbb{Q}} \ge 2$, the remaining exponent slack absorbs all logarithmic factors. The initializer and runtime lemmas apply to the combined banks, giving $O(\log n)$

precision and total depth $O(\log^2 n) + O(\log n) = O(\log^2 n)$. Empty input is handled by the fixed grammar's empty-word membership bit. □

B.8 Shared initialization and normalized execution

The blank constructions in Sections B.5.3, B.6.3 and B.7.3 use three shared guarantees: static fields can be generated and released synchronously (Lemma B.18); execution preserves address reads and discrete decisions (Lemma B.19); and local Boolean circuits can be generated and evaluated internally (Lemma B.20). Each application supplies its layout decoder and storage count. The local circuits serve UCFL and general-CFL arithmetic; DCFL uses unique reads and finite-state updates. Public implementations use the same execution rules with supplied static fields.

We first build fixed control, recover positions, and implement word operations. A common clock and canonical keys then complete initialization, after which we prove the execution and circuit guarantees. All modules have fixed parameters and constant width, and initialization counts toward the blank loop bounds. Here $N = n + 2$; padded matrix sizes are specified separately in the applications.

B.8.1 Fixed control and numerical representation

A fixed repeated block must select its own instructions, leave disabled writes exactly zero, and reuse numerical buffers without accumulating uncontrolled error. We implement this control for a fixed finite program with constantly many word buffers. The representation below separates exact finite states from numerical directions with controlled rounding error.

Numerical and architectural conventions. We use the multi-pre-norm convention of Jerad et al. (2026, Appendix A): a fixed number of rational projections are normalized separately and concatenated before fixed linear or ReLU maps. In particular, finite control and numerical blocks need not share a normalization factor, and a head's value projection can isolate a source block independently of its scoring fields. The number of projections is constant, so width and block depth remain constant. Normalize mean-zero vectors by their Euclidean norm. The usual variance normalization differs by a fixed factor; padding the ambient dimension to a perfect square makes its compensation rational. There is no additive numerical epsilon. Use nearest dyadic rounding R_p on a grid of spacing $\tau = 2^{-p}$, with midpoint ties toward the higher value, as permitted by the precision convention in Merrill & Sabharwal (2025a, Appendix A). A fixed number of integer bits covers every stored or projected scalar. Parameters do not depend on p or n. Empty strictly causal attention at BoS returns zero; BoS's immutable symbol fields are retained by residual connections.

Midpoint rounding need not commute with negation. For a balanced vector $v = (a, -a)$ define the paired normalization

$$z = R_p\big(v/\|v\|\big), \qquad \mathsf{P}(v) = (z_+ - z_-, z_- - z_+). \tag{27}$$

This is normalization followed by a fixed linear map. Its output is exactly balanced, differs from $2v/\|v\|$ by $O(\tau)$ in fixed dimension, and has norm $2+O(\tau)$. All subtractions are exact grid operations. We store numerical words and address directions at ideal radius two. Boolean and finite control states instead use balanced one-hot codes with four nonzero coordinates $1/2, 1/2, -1/2, -1/2$, of exact unit norm. These conventions avoid any sign-symmetric rounding assumption.

Lemma B.13 (Fixed control of normalized instructions)**.** *Consider a fixed finite program over constantly many paired projective word blocks with uniformly bounded residual coordinates. Its instructions apply fixed rational affine maps, test affine guards, copy blocks, and branch on finite control. Assume active normalization inputs are nonzero and each active guard's error is smaller than its ideal distance from zero. At sufficiently fine precision, one repeated constant-depth pre-norm/ReLU block implements the program with exact finite control and exactly zero inactive writes, including from zero source buffers. Normalization and readout introduce $O(\tau)$ fresh rounding error per write; two-buffer reuse leaves $O(T\tau)$ clearing residue after T writes. Propagation of numerical input error is accounted for separately in Lemmas B.15 and B.19.*

Proof. (i) Enabled maps and guards. Represent a tuple x by the paired direction of $(x, 1, -x, -1)$, at ideal radius two. A fixed affine map is a homogeneous matrix H. Multiply all its coefficients by a

fixed positive common denominator, which leaves its normalized direction unchanged. Its projection can therefore use integer coefficients and is exact on the chosen grid within a fixed scalar range.

For an exact enable bit b, append a balanced gate pair $d, -d$ to Hu, where $d = D(1-b)$ and the fixed integer D exceeds all data-coordinate magnitudes in the maps and guards on the bounded residual blocks. Normalize and round this whole vector to (z, g_+, g_-), and set $g = g_+$. The rounded gate coordinates need not be exact negatives. Apply, coordinatewise,

$$t_j = \mathrm{ReLU}(z_j - g) - \mathrm{ReLU}(-z_j - g), \qquad \text{output } (t_+ - t_-, t_- - t_+). \tag{28}$$

When $b = 1$, $g = 0$ and this is precisely $\mathsf{P}(Hu)$. When $b = 0$, the gate coordinate strictly dominates all data coordinates by a constant margin after normalization; thus every t_j is exactly zero at sufficient precision. The inactive projection is nonzero because $d = D$. The enable and constant coordinates come from exact finite-control and immutable marker fields through a fixed projection.

For a guard Δ, normalize $(\Delta, \Delta, -\Delta, -\Delta, D(1-b), -D(1-b))$. Again let g be the rounded positive gate coordinate. The readout $2\,\mathrm{ReLU}(z_1 - g)$ is exactly zero when disabled. When enabled, the first four normalized coordinates are $\pm 1/2$, exactly representable on the grid, so the readout is $\mathbf{1}[\Delta > 0]$. The guard-error assumption preserves the ideal sign; the later word-operation and runtime lemmas verify this assumption for the applications.

(ii) Exact finite control. Concatenate the balanced one-hot control blocks, padding with fixed unit blocks until their number is a power of four. Their common normalization is multiplication by a dyadic constant, so all control bits and an immutable constant are recovered by fixed rational weights. AND is $\mathrm{ReLU}(a + b - 1)$ on bits; complements and exclusive sums implement a finite transition table. Subtract the old control code and add the new one by a residual update. Thus control remains exact by induction.

(iii) Fixed instruction schedule. For every possible program edge and each source-buffer parity, include a fixed instance of (28). Its enable is the AND of the current state, parity, and guard outcome. Evaluate all instances every iteration; exactly one enabled instance writes its destination. Old control and source fields remain frozen until these sublayers finish. A final finite update changes the state and buffer parity. There are only constantly many states and matrices, independent of input length.

(iv) Buffer reuse. Write the new normalized value into the previously cleared buffer. The normalized readout supplies $\mathsf{P}(u)$ rather than the raw old buffer u, so clearing by subtraction leaves $u - \mathsf{P}(u)$. Let E_t bound previous clearing residues, and let $\delta = O(\tau)$ bound both fresh normalization error and the new value's norm error. The old buffer contains such a value plus a prior residue, hence $|\|u\| - 2| \le E_t + \delta$. Since $\|u - 2u/\|u\|\| = |\|u\| - 2|$, the next residue satisfies

$$E_{t+1} \le E_t + 2\delta.$$

Thus clearing contributes $O(T\tau)$ over T writes. Disabled writes are exactly zero, so idle steps add no residue. This bound concerns clearing, not propagation of errors through the computed directions. Initial rational embeddings seed the buffers with constantly many normalized instructions. □

Head reads and conditional writes. A head-facing direction field is the fixed sum of its two buffers. One holds the active block and the other has $O(T\tau)$ clearing residue, which is the additional error in this readout. Identical complete two-buffer traces give identical readouts; Lemma B.17 establishes this condition for equal-label static keys. Commit an update before a later head reads the field; separate snapshots retain the old values needed by synchronous pointer phases.

An attention write can also be disabled exactly. For an exact enable bit b, add $B(2b-1)$ to every non-BoS key's score and zero to BoS's score, where the fixed rational B exceeds the magnitude of the remaining score. This is a query control coordinate paired with a finite non-BoS key coordinate. BoS has zero base score and zero value. When $b = 0$ it is the unique maximizer and the head writes exactly zero; when $b = 1$ it loses, and the intended maximizers among earlier non-BoS records are unchanged. Each active query has an earlier matching default or genuine source. For a unique direction read, normalize the paired source block separately and apply the paired linear readout. Its value has norm $2 + O(\tau)$. Use one fixed head per destination-buffer parity, write into the cleared destination, and clear the old buffer before committing. The preceding residue bound therefore applies to these copies. A balanced BoS-only marker keeps the value projection nonzero at BoS; the linear value readout discards it. Direction buffers start from a rational radius-two default, for example

$(1, 1, -1, -1)$ padded with zeros, with the other buffer zero. The first seed write replaces this default, whose clearing is exact.

Two other writes use separate storage rules. A read whose maximizers all carry the same finite code returns that code exactly: normalize only its finite value block, with fixed padding as in part (ii). Its scratch can be cleared by subtracting the exactly decoded old code before the next read. In contrast, a numerical mean need not have radius two. Write each such mean only once into dedicated initially zero scratch, then disable that head permanently for the evaluation. Subsequent guards may read its direction, but neither clear nor overwrite the raw mean. Bootstrap reciprocals and ready-gate aggregates follow this rule (Lemma B.20), with constant scratch and one written flag per record. The numerical clearing bound applies only to normalized directions.

B.8.2 Position recovery and word reconstruction

Lemma B.14 (Causal bootstrap and a common template)**.** *Suppose the input precedes polynomially many identical blank tokens, with an immutable BoS marker and no supplied positional fields. Under the conventions of Section B.8.1, $O(\log n)$ controller instructions recover scaled position and length words with inverse-polynomial error, and a protected length-dependent template whose stored trace is bit-identical at every blank. Input positions obtain frozen address keys for unique symbol retrieval. Width is constant and sufficiently large $O(\log n)$ precision preserves all bootstrap decisions. The template alone does not yet supply canonical keys for arbitrary labels.*

Proof. We describe ideal register values first; finite-precision execution follows the same branches and approximates these values, as justified below.

(i) Reciprocal positions and input keys. Index BoS by zero and let $m = n + 1$. On a blank at position i, a uniform strictly causal head with value one only at BoS gives $r_i = 1/i$ before rounding. A second head prioritizes BoS and input-symbol keys over blank keys, giving $r_m = 1/m$. Key scores depend only on immutable symbol fields; both heads retain the strictly causal mask. The bootstrap value comes from a separate exact Boolean at BoS, initially one. After these two heads execute, a fixed local startup transition sets it to zero permanently. Thus the heads write their means once into fresh scratch and subsequently write zero, including at the first input position whose only earlier record is BoS. All blanks receive bit-identical copies of r_m. Input records need no knowledge of n: the first head also gives their own reciprocal position. In a fixed bootstrap prefix they store $\mathsf{P}(1, r_i, -1, -r_i)$ as an input-position key and then freeze it. This approximates the direction of $(i, 1, -i, -1)$ to inverse-polynomial accuracy and supports unique symbol retrieval after initialization. Fixed endmarkers change m by only a constant.

(ii) A protected common template. Starting with $u = 1$, halve while $3u > 2r_m$. If $u = 2^{-q} = 1/Q$ at termination, then $3m/2 < Q < 3m$. The guard cannot be zero because $3m$ is not a power of two. A fixed number B of repetitions of this halving count generates $\sigma = Q^{-B}$. Compute and freeze a projective template for $(0, \sigma)$ in a block whose projections exclude r_i and all local work. Its entire trace is therefore identical at every blank. Choose fixed B sufficiently large that all initializer integers and intermediates remain bounded after scaling, and σ is no larger than the local position scale u constructed next.

(iii) Scaled position and length words. To recover a position, start with $u = 1, v = r_i$ and repeatedly set $u \leftarrow u/2$, $v \leftarrow 2v$ while $3u > 2r_i$. Initialize $h = 1, a = b = 0$. While $4h > 3u$, add h to a and v to b if $2 + r_i > 2b + 2v$, and then halve h, v. Throughout this scan,

$$b = \frac{a}{iu}, \qquad v = \frac{h}{iu}.$$

Thus the test accepts the candidate integer $j = (a + h)/u$ exactly when $j < i + 1/2$, or equivalently $j \le i$. The scan tries descending binary places through $h = u$, producing $a = iu$. Finally halve a, u until $2u < 3\sigma$, using a local scale generated by the same length-only program. Since $u \ge \sigma$ and both ideal scales are powers of two, this stops at $u = \sigma$ and $a = i\sigma$. The same procedure recovers $m\sigma$; subtracting σ gives $n\sigma$.

For polynomially many positions, the program uses $O(\log n)$ instructions on bounded tuples. Numerical updates preserve a homogeneous anchor, so their normalization denominators are bounded away from zero and each has a constant Lipschitz bound. Over $O(\log n)$ steps, amplification is

polynomial. Ideal guard values are nonzero rationals with polynomially bounded denominators: the halving stops separate powers of two, and the digit test uses a half-integer threshold. Their margins are therefore inverse-polynomial. Taking $p = C\log(n+2) + O(1)$ with sufficiently large fixed C keeps accumulated error below every margin and preserves all branches. This also bounds bootstrap and template error by a polynomial multiple of 2^{-p}. Subsequent integer reconstruction uses the protected common template, avoiding dependence on the local scale's numerical history. □

The recovered words seed the layout decoder. Its integer operations use bit-serial reconstruction, consuming bits sequentially in constantly many word buffers.

Lemma B.15 (Word operations with reconstruction)**.** *Given the bootstrap words and protected template of Lemma B.14, consider a fixed constant-word program with constant call depth and $O(\log n)$ operations drawn from addition, subtraction, multiplication, quotient/remainder with nonzero divisors, comparison, bounded shifts, and bit access. Assume all integers and intermediates are polynomially bounded, with $O(\log n)$-bit scans and shift counts of $O(\log n)$. The program produces exact output bits and reconstructed numerical words in $O(\log^2 n)$ controller instructions, at $O(\log n)$ cost per operation. Reconstruction resets numerical error to a common polynomial multiple of 2^{-p}, independent of preceding numerical history; one fixed $p = O(\log n)$ suffices throughout.*

Proof. (i) Integer operations. A working block stores the paired direction of

$$(x_1\sigma, \dots, x_r\sigma, \sigma, 1),$$

where r is fixed; its operands, scale, and homogeneous anchor share one normalization factor. Choose the template exponent so the scaled integers and arithmetic intermediates have magnitude at most one. Guards and updates operate within this tuple: for example, comparing x_1 and x_2 uses $x_1\sigma - x_2\sigma - \sigma/2$, and a Horner step uses $z\sigma \leftarrow 2z\sigma + b\sigma$. All coefficients are fixed rational numbers, and the anchor implements affine constants.

To move a word between independently normalized blocks, transfer its bits, not its numerical coordinates. Copy the source tuple to a scan buffer, retain its own scale and anchor, and test $x\sigma + \sigma/2 < 0$ for a negative integer. Negate that operand when the test succeeds. For its magnitude, double a cursor from σ past $3/2$, then descend through the binary places. With remainder a, the test $a - h + \sigma/2 > 0$ decides whether to subtract the current place h and emit one. Stop after processing $h = \sigma$, detected by $2h < 3\sigma$. The ideal tests have nonzero half-grid margins. All scan registers share the source tuple's normalization; the stored scale need not be exact.

Seed a destination tuple from the protected template with zero operand slots. As each exact source bit arrives, apply a Horner step to its destination slot, using the destination's own scale. Repeat for the fixed number of operands while retaining previously reconstructed slots. This uses $O(\log n)$ instructions and places all operands under one common factor without comparing coordinates from different blocks. Multiplication then scans multiplier bits with updates $z \mapsto 2z$ or $z \mapsto 2z + y$. Long division scans dividend bits, updates $r \mapsto 2r + b$, conditionally subtracts the divisor, and emits a quotient bit; in the working tuple the incoming bit contributes $b\sigma$. Addition, subtraction, comparison, bounded shifts, and bit access similarly use $O(\log n)$ affine/guard instructions, with signs in finite control. Each operation ends by decoding and reconstructing its result from the protected template. The scans and updates use a fixed number of word buffers and the finite controller of Lemma B.13.

(ii) Reconstruction and error induction. Write $N = n + 2$. Numerical updates preserve the homogeneous anchor of the bounded scaled tuple, keeping their normalization denominators uniformly away from zero. Each fixed affine normalized update therefore has a constant Lipschitz bound, and a logarithmic-length segment amplifies error by at most $K^{c\log N} = N^{b_1}$ for fixed b_1. Guard normalization is handled separately: half-grid comparisons produce exact bits when their input error is below the nonzero margin. Their outputs carry no analog error into the next update. Buffer reuse adds $O(T\tau)$ at each write. The initial bootstrap and protected template have error at most $N^{b_2}\tau$. A reconstruction from correct bits therefore has a fixed bound $R(N,T)\tau$, for one polynomial R, independent of the numerical history that produced those bits. If $A(N)$ bounds the next operation and decoding amplification and N^{-g} is a lower bound on guard margins, choose precision so that

$$A(N)R(N,T)\tau < N^{-g}/4. \tag{29}$$

Under (29), the output bits are exact, and reconstruction returns the error to $R(N,T)\tau$. Induction over all word operations uses this same bound; it does not multiply $A(N)$ once per word operation. Thus $p = C\log N + O(1)$ with a fixed sufficiently large C suffices for $T = O(\log^2 N)$. □

B.8.3 CANONICAL FIELDS AND SYNCHRONIZED INITIALIZATION

Local decoders may finish at different times. A common timer starts the canonical-key phase after all local work is complete; a fixed-length key scan then permits a common transition to execution. Lemma B.18 combines these steps into the interface used by the parsing constructions.

Lemma B.16 (Common clock and internal barrier)**.** *At the end of the common length-dependent bootstrap phase, let every blank have identical protected length fields and a bound Aq^2 on its remaining local work, where A is fixed and $q = \Theta(\log(n+2))$ is the common halving count. A fixed controller starts canonicalization at every blank on one common tick after that work, using $O(q^2)$ instructions, constant width, and $O(\log n)$ precision. It reads no other position's completion flag. Early finishers idle their local decoder while the timer continues.*

Proof. All blanks start dedicated timer fields at the same transition out of the length-dependent bootstrap phase. The timer advances in fixed sublayers of each repeated block, independently of local branches or completion. The common halving count gives q. Two counters x, y count q steps each, resetting x when it reaches q and incrementing y. Repeat this rectangle a fixed number of times. Tests use $q - x - 1/2$ and $q - y - 1/2$; updates are increments, resets, and copies with fixed finite control.

Store the timer tuple (q, x, y) projectively with a homogeneous unit anchor. Its decoded coordinates are $O(q)$, so the normalized anchor is $\Omega((q+1)^{-1})$. Converting a perturbation of the stored direction to decoded coordinates amplifies it by at most $O((q+1)^2)$. On a fixed branch, the decoded update is nonexpansive in the maximum norm. Let e_t be the decoded error and $T = O(q^2)$ the timer duration. Fresh rounding and the buffer residue from Lemma B.13 give

$$e_{t+1} \le e_t + C(q+1)^2(T+1)\tau, \qquad e_T \le e_0 + CT(T+1)(q+1)^2\tau \tag{30}$$

for a fixed C. The count q is accumulated from a fixed rational seed in $O(q)$ increments, giving $e_0 = \mathrm{poly}(q)\tau$ by the same estimate. Choose precision so this bound is below $1/8$. Every ideal guard is a half-integer, and its error is at most $2e_t$, so induction preserves all branches. Thus logarithmic precision suffices throughout the quadratic-logarithmic timer, without chaining the amplification bound for general word operations.

Choose the fixed repeat count to cover Aq^2 local instructions and the constant scheduling overhead. Once local work finishes, its writes are exactly disabled. Timer fields and control depend only on common initial fields and receive the same updates at every blank, so the transition to canonicalization is bit-identical across records. □

Lemma B.17 (Canonical keys for equal labels)**.** *Given polynomially bounded label words satisfying Lemma B.15, the protected common template, and a synchronized start, a fixed $O(\log n)$-instruction phase generates dedicated key buffers. Equal labels have bit-identical stored keys, including the two-buffer readout; distinct labels retain inverse-polynomial address separation at sufficiently large logarithmic precision. Local scratch and dynamic payloads are excluded from key generation.*

Proof. After the common barrier, initialize both dedicated key buffers from the same fixed defaults and protected template, without reusing decoder scratch. For each of constantly many label components, the sequential bit-decoding procedure of Lemma B.15 supplies exact bits in one $O(\log n)$ scan. Use a common length determined by q, padded with leading zeros, and the same fixed number of instructions per bit. The scan's numerical scratch is excluded from the key projections. Equal labels therefore drive identical complete two-buffer traces and produce bit-identical readouts under the specified rounding rule.

For each canonical integer direction, apply the homogeneous Horner map as each bit b arrives:

$$H_b = \begin{pmatrix} 2 & b \\ 0 & 1 \end{pmatrix}, \qquad b \in \{0, 1\}. \tag{31}$$

Both matrices have fixed positive smallest singular values, so their normalized maps have constant Lipschitz bounds and the $O(\log n)$-step trace has polynomial error amplification. For polynomially bounded labels, choose precision so the resulting score perturbation is smaller than a fixed fraction of the gap in (13). Distinct labels remain separated, while equal-label ties follow from identical traces. The scans finish together, and a common finite transition enables execution. □

Lemma B.18 (Uniform word initialization and an internal barrier)**.** *Suppose a blank record's constantly many static fields are decoded from (i, n) by a fixed constant-word program with constant call depth and $O(\log n)$ integer operations satisfying Lemma B.15. Such fields, including canonical directions and fixed inverse-polynomial scales, can be generated from identical blank padding in $O(\log^2 n)$ loops, constant width, and $O(\log n)$ precision. All blank records then switch to execution synchronously, without an external schedule.*

Proof. The common length-dependent phase of Lemma B.14 supplies the protected template and starts the timer of Lemma B.16. Remaining position recovery and the static-field decoder run alongside it; Lemmas B.14 and B.15 bound their total cost by Aq^2 for a fixed A. At the common barrier all blank records enter Lemma B.17's fixed-length key-generation phase and then start execution together. The phases use constant width and total $O(\log^2 n)$ depth. Taking the largest required precision constant gives one $O(\log n)$ precision bound. □

B.8.4 Reliable execution at logarithmic precision

Execution needs three guarantees: unique reads select the intended source, grouped reads preserve exact ties, and rounded aggregates yield exact discrete decisions. The following lemma separates these guarantees.

Lemma B.19 (Runtime precision for reads and aggregation)**.** *Suppose static fields satisfy the canonical-key and numerical-error guarantees of Lemmas B.17 and B.18, either supplied publicly or generated internally. Consider at most $T = O(\log^2(n + 2))$ steps of direction copying and finite-state updates. Comparisons have fixed depth and polynomial error amplification, and return exact bits; their numerical scratch is not propagated. Labels are polynomially bounded, and score and guard gaps are inverse-polynomial. Unique reads may use controlled-error directions. Tied reads use canonical group keys and equal-norm finite codes related by coordinate permutations, with identical marker and other retained fields within each group. Varying source addresses and scratch are excluded from the aggregation projections. Finite-code scratch is reused exactly; each numerical mean is written once into separate zero scratch and retained, as in Section B.8.1. Under these conventions, one fixed $p = O(\log n)$ preserves unique selection, intended ties, and exact Boolean/finite decisions. Reused-buffer direction copies incur $O(T^2 2^{-p})$ additional error. Integer-vote thresholds use a common positive marker and the layout's known reserved fan-in, including neutral and bias slots; means may be rounded.*

Proof. (i) Unique reads and copied directions. Store numerical address components as paired directions and refresh finite payloads to exact codes. Section B.2.1 gives the inverse-polynomial score separation for polynomially bounded labels. Unique reads copy directions through the paired normalization of Lemma B.13. With fresh buffers, T copies introduce $O(T\tau)$ error; reused-buffer residues give $O(T^2\tau)$. Indeed, normalizing a block of radius $2 + O(T\tau)$ has local Lipschitz factor $1 + O(T\tau)$ in direction coordinates, so

$$e_{t+1} \le (1 + O(T\tau))e_t + O(T\tau), \qquad e_T \le (1 + O(T^2\tau))e_0 + O(T^2\tau) \tag{32}$$

when $T^2\tau$ is small. Here e_0 is the initialization error; the last term is the additional copying error. This bound applies only to normalized direction copies, not to raw attention means. Fixed-depth comparisons amplify this bound by at most a polynomial factor. Keeping it below their guard margins preserves exact output bits, and their numerical scratch does not continue along the copying chain.

(ii) Canonical tied aggregation. Within a group, canonical key fields, markers, and other retained fields are bit-identical. Payload codes occupy a separate block and differ only by coordinate permutations, so the whole projected vectors have identical norms. Their normalized key coordinates are therefore identical, and coordinatewise rounding preserves this equality. The finite payload and marker coordinates also undergo the same rounding in every row. Thus the intended group ties exactly, while the assumed score gap excludes other records. No sign symmetry of rounding is required. Repeated existence queries instead rank a requested group first and an exact predicate bit second, using an initialized inverse-polynomial priority smaller than the group gap. Include a predicate-zero default. All maximizers then have the same predicate bit, so a separately normalized finite value block returns it exactly, even when several records tie. Such queries use the exact finite-code scratch protocol of Section B.8.1.

(iii) Integer thresholds and precision. Let d be the reserved number of rows in a nonempty aggregation group, including neutral slots and any bias slot. Genuine edges supply votes $2b - 1$ and nonedges supply zero. Strict majority tests positivity of the signed sum s; non-strict majority adds a unit bias before that test. Choose the marker equal to a positive one-hot coordinate and read votes as differences of positive one-hot coordinates. By part (ii), these positive coordinates have one common stored value $\lambda > 0$ after source normalization and rounding. Before rounding the means,

$$a = \lambda s/d, \qquad h = \lambda, \qquad 2da - h = \lambda(2s - 1). \tag{33}$$

Since s is integral, this expression is positive exactly when $s \geq 1$ and has magnitude at least λ. No division by the common scale is needed. If each rounded mean has error at most η, then

$$\left|2d\widehat{a} - \widehat{h} - (2da - h)\right| \leq (2d + 1)\eta.$$

Taking $(2d + 1)\eta < \lambda/2$ preserves the sign. The means are written once, so their error contains no residue from clearing an earlier mean. A fixed-depth comparison latches the resulting exact bit and leaves the numerical scratch unchanged.

The reserved d and two earlier comparison anchors are static layout fields, supplied publicly or generated by Lemma B.18. Address and role priorities select the matching anchor pair; its comparison directions approximate $\pm(2d, -1)$. Paired key readouts from a common canonical direction give exactly opposite comparison components under the specified rounding rule. The factor d thus enters through an initialized key, not a length-dependent weight. Since $|s| \leq d$, normalizing the ideal query (a, h) and anchors leaves a score gap $\Omega(1/d)$ before the common priority scaling. Both d and λ^{-1} are polynomially bounded, and the priority scale is inverse-polynomial. A sufficiently large precision constant keeps mean-rounding and anchor errors below the gap. Together with initialization and parts (i)–(ii), this gives $p = O(\log n)$ throughout execution. □

B.8.5 Local Boolean circuits

Lemma B.20 (Local Boolean circuits with public or blank initialization)**.** *Fix a DLOGTIME-uniform family of depth-D_0 circuits over Boolean and majority gates, where $D_0 \geq 1$ is constant and size $S = C_0 R^C$ includes all bit ports, and let R be polynomially bounded in n. Each input port reads an exact bit and ready flag from an earlier bank; a ready input remains fixed. Each allocated module evaluates once.*

1. *Under public initialization, suppose module locations, sizes, and external bit-port labels are logspace computable. The module uses $O(S^2)$ records and returns exact output bits and ready flags in $O(D_0)$ loops once its inputs are ready.*
2. *If these labels admit the decoders of Lemma B.18, polynomially many such modules can instead be initialized from blank padding in parallel within $O(\log^2 n)$ loops, with the same storage and subsequent execution bounds.*

In both cases width is constant and precision is logarithmic. A causal module DAG of depth $H = O(\log^2 n)$ executes in $O(H)$ loops once its external inputs are ready; blank initialization is charged once. Arithmetic results are returned as exact bits. Address-direction conversion and any output rescaling are supplied by the calling construction.

Proof. (i) Layout, gate, and port decoding. Reserve $D_0 + 2$ gate copies, numbered $0, \ldots, D_0 + 1$, and S^2 potential edge slots between adjacent copies. The extra copy permits a zero-input internal gate to become ready in one execution round, even when circuit depth counts that gate as depth zero. Within a module, fixed-stride blocks encode the copy index, record role, and gate or source–target pair. Given a physical position, the outer decoder identifies its module and offset; quotient and remainder then recover these local indices. Thus locating a record is separate from deciding whether its potential edge is genuine.

Direct-connection, gate-type, and input/output-port predicates take $O(\log R)$ random-access Turing steps. For public initialization, the logspace outer label procedure and these predicates enumerate all roles, edges, and ports. Equal labels receive identical rounded fields by simulating the fixed canonical reconstruction trace of Lemma B.17, using $O(\log n)$ workspace. Rounding fixed-dimensional

normalized vectors reduces to integer comparisons at the chosen dyadic precision. Thus the static fields are generated externally without initialization loops.

For blank initialization, the predicates' tapes and query indices fit in constantly many $O(\log n)$-bit words, with each transition simulated by constantly many word operations from Lemma B.15. The outer decoder resolves external operands and fixed coefficient bits. Together these procedures satisfy Lemma B.18 for roles, edges, ports, and auxiliary fields. Unused identifiers and padded ports are inactive; the quadratic reservation is local to each module.

(ii) Gate semantics and readiness. After initialization, constants are ready. Each input port reads its own earlier source bit and ready flag, retaining the bit once ready. Internal gates start unready; copy zero supplies only constants and input ports. In later copies, a gate evaluates only after every genuine predecessor is ready, or carries forward its own ready value from the preceding copy. Ready values never change during the evaluation.

An edge record latches its source bit when its source becomes ready. A waiting gate repeatedly asks whether any genuine incoming edge is still unready, using the finite-valued existence query of Lemma B.19. Nonedges and the matching default have predicate zero. Every maximizer carries the same exact predicate bit, so repeated readiness checks require no numerical mean buffer. Once the answer is zero, all genuine edge payloads are fixed. The gate enables its value head once, writes to initially zero scratch, performs the threshold test, and latches the output bit and ready flag. A written flag prevents a second value write; the scratch is retained thereafter.

Use balanced one-hot codes for votes $-1, 0, +1$. Genuine edges supply $2b-1$ for majority; an absent edge uses the neutral code, whose vote readout is zero but whose norm and marker match the other codes. Strict majority tests positivity; non-strict majority adds the unit bias of Lemma B.19. OR tests for a genuine input equal to one, and AND negates the test for a genuine zero. NOT complements its sole input once that input is ready. Nonedges do not delay readiness, and empty OR and AND evaluate to zero and one, respectively, in the first eligible round. A neutral dummy slot makes every value aggregation nonempty; it and any bias slot retain the same marker. All thresholds use the known reserved fan-in, including these auxiliary slots.

(iii) Causal reads and tied aggregation. Place gate copy j before its outgoing edge records and these before gate copy $j+1$. An edge reads its source bit and readiness in copy j; the next gate copy aggregates those earlier edges or carries forward its own completed value from the preceding copy. Matching defaults and comparison anchors precede their readers. A fixed cycle first reads and latches source bits at edges, then checks gate readiness, writes enabled value aggregates, compares them, and commits gate outputs. These are a constant number of stages per logical round. Carried values and gates already ready bypass aggregation; all their numerical writes are disabled.

Source bits use unique reads. For each target and aggregation type, projections retain only canonical group labels, finite payloads, and common marker/role fields. The payload codes are coordinate permutations of one equal-norm code; all other retained fields agree within a group. Varying source addresses, coefficient magnitudes, and scratch are excluded. These are precisely the tie and threshold conditions of Lemmas B.17 and B.19.

(iv) Correctness and resources. A ready gate has the correct value: its genuine predecessors were ready and correct when it latched, or it copied its own correct ready value. A zero-input gate uses its prescribed Boolean value. Measure depth by the longest input-edge path, so constants, input ports, and zero-input gates have depth zero. After the external inputs are ready and one port-read stage has completed, all port copies are ready. Copy one therefore contains all depth-zero values after one logical round. Inductively, copy j contains correct ready values for every gate of depth at most $j-1$ after j rounds: its predecessors of depth at most $j-2$ are ready in copy $j-1$, and the next round reads their edges before evaluating the gate. Constants and ports persist, and completed values can be carried forward. Copy D_0+1 thus contains every output after D_0+1 rounds. A module finishes in $O(D_0)$ loops, including port reads. Evaluating independent modules in parallel gives $O(D_0 H) = O(H)$ loops for a DAG of depth H, since D_0 is fixed.

The public case begins with these fields supplied; the blank case uses Lemma B.18 to initialize every module bank in parallel in $O(\log^2 n)$ loops. The gate copies and edge copies, together with ports, defaults, dummy and bias slots, and comparison anchors, occupy $O((D_0+1)S^2) = O(S^2)$ records per module. Each gate needs only constantly many retained mean coordinates and exact control bits;

no scratch is allocated per waiting round. Their reads and decisions satisfy Lemma B.19, so one fixed logarithmic-precision constant covers initialization and the full module DAG. □

B.9 Resource accounting and exponent conventions

Public initialization supplies fields determined by the input length and fixed formal system. Their records count toward $M(n)$; their external generation is excluded from $L(n)$. We distinguish this generation cost from execution work below. Let $R(n) = O(M(n))$ count public and working records in the chosen construction.

Public field generation. A fixed generator uses $O(\log n)$ workspace and emits $O(R(n) \log n)$ bits. Record labels specify interval endpoints, grammar or automaton states, stages, local gate/edge labels, and signed relay paths. Following these paths identifies source labels. Coefficients are emitted as bits; canonical directions follow the fixed rounded reconstruction trace of Lemma B.20. Each trace uses logarithmic-bit integer arithmetic and polylogarithmic time, including normalization rounding by integer comparisons. A conservative generator repeatedly scans legal labels to emit the next record in causal order and to count those preceding each of its constantly many source labels. Keeping only constantly many labels and counters gives logarithmic workspace and $\widetilde{O}(R(n)^2)$ bit operations, where the tilde suppresses polylogarithmic factors. The generator ignores input symbols, so its output can be reused for all words of the same length and formal system.

Remark B.21 (Generic initialization versus direct decoding). The public serialization is a polynomial-output logspace transduction. The generic causal simulation of Merrill & Sabharwal (2025a, Definition 10 and Lemmas 3–4) provides polynomial storage, but does not guarantee the target storage exponents after output and packing. The blank constructions therefore use the direct decoders in Sections B.5.3, B.6.3 and B.7.3.

Dense attention work. With constant width, head count, and layers per loop, dense strictly causal attention evaluates $\Theta(M(n)^2)$ score pairs per loop, giving $O(L(n)M(n)^2)$ scalar work. Operations on logarithmic-bit fields are counted at unit cost; bit-operation accounting adds polylogarithmic factors. For example, the DCFL bounds give $O(n^4 \log^2 n)$ dense work, whereas classical Cocke–Kasami–Younger (CKY) parsing uses $O_G(n^3)$ work and $O_G(n^2)$ chart space. These storage and loop-depth bounds do not establish a work or runtime improvement over CKY.

Comparison with full-layer copies. Consider a simulator that stores the initial state and each of L simulated layers in separate blocks of W records. If every record in layer $t+1$ directly reads every record in layer t, strict causality requires $\max P_t < \min P_{t+1}$ for their position sets. This layout therefore uses $W(L+1)$ positions.

The derived DCFL baseline in Table 1 applies the full-layer simulation of Merrill & Sabharwal (2025a, Lemma 1 and Appendix C) to the public AHAT realization in Section B.5 of Chytil et al. (1991)'s interval-summary framework, including its witness fields. With n input records, source padding $p = O(n^2)$, and depth $L = O(\log^2 n)$, the copied layout uses $p + L(n+p) = O(n^2 \log^2 n)$ padding. Our DCFL construction instead reuses each interval's work bank throughout its merge; these banks retain child summaries and total $O(n^2)$ records.

Exact rational rank and exponent conventions. The coefficient field in $\omega_{\mathbb{Q}}$ specifies the exact identities executed by our circuits. This exponent agrees with the usual characteristic-zero matrix-multiplication exponent. A complex rank-r identity of side m satisfies polynomial coefficient equations over $\mathbb{Q}$, so such an identity also exists over $\overline{\mathbb{Q}}$. Its coefficients lie in a fixed finite extension $E/\mathbb{Q}$. Expressing arithmetic in a rational basis after t tensor powers gives a rational identity of rank at most $C_E r^t$ for side m^t, with C_E independent of t. Taking $t \to \infty$ removes this constant from the exponent; the reverse inequality follows from $\mathbb{Q} \subset \mathbb{C}$. Rank and border rank have the same asymptotic exponent by interpolation after tensor powering (Bini, 1980). The ε slack lets us choose a fixed exact rational identity before n varies, as required by the arithmetic constructions above.

C STORAGE AND REPRESENTATION LIMITS OF SPECIFIC INTERFACES

A fixed grammar has only $O(n^2)$ chart items $A[i, j)$ on a length-n input, one for each nonterminal and substring. Storing the final chart takes quadratic space, but computing it may require additional records for intermediate dependencies. For unambiguous context-free parsing, our causal AHAT constructions use subcubic storage with constant width, $O(\log n)$-bit precision, and $O(\log^2 n)$ loops.

We examine two approaches to quadratic-storage AHAT parsing: explicitly recording dependencies, and representing chart membership directly through attention scores. The first can create cubically many records even for an unambiguous grammar. The second can require score dimension or head count to grow with input length.

C.1 EXPLICIT DEPENDENCY RECORDS

Even for a reduced unambiguous grammar, unique successful joins do not guarantee a quadratic number of dependency records.

Transitive dependencies. For the fixed unambiguous grammar $C \to TC \mid a$, $T \to a$, the word a^n admits a context $C[i, j) \Rightarrow^\star a^{k-i}C[k, j)$ for every $0 \le i < k < j \le n$. This context summarizes a chain of deductions with $C[k, j)$ left to fill: deriving that item suffices to derive $C[i, j)$. There are $\binom{n+1}{3} = \Theta(n^3)$ such root–hole pairs, so assigning a separate shortcut record to every pair takes cubic space, although the chart itself has only quadratic size.

Unsuccessful candidates. Consider the fixed unambiguous grammar $S \to BC$, $B \to b$, $C \to TC \mid c$, $T \to c$ on the negative word c^n. Once all realizable C items are known, the one-known-sibling relation contains an edge $S[i, j) \to B[i, k)$ for every $0 \le i < k < j \le n$, since $C[k, j)$ is true. These $\binom{n+1}{3}$ candidate edges all lead to false B items, and no S join succeeds. Thus the candidate relation for S is cubic despite having no successful joins. Sharing the rejection of each B item can avoid storing its incident edges.

Independent-query bound. In a separate oracle model, suppose candidate truth values are independent bits. Deciding exactly whether any of $\Theta(n^3)$ candidates is true requires inspecting every bit on the all-zero transcript. With $M(n)$ work records, $L(n)$ rounds, and $O(1)$ queries per record per round, $M(n)L(n) = \Omega(n^3)$. If each record's $O(1)$ query addresses stay fixed throughout execution, $M(n) = \Omega(n^3)$. The bound applies to independent bits; grammar-induced candidates share factors that an implementation can reuse.

C.2 DIRECT ATTENTION MASKS

Another approach is to encode the chart directly in attention scores. For a fixed nonterminal, regard substring starts as rows and ends as columns. A direct mask requires the keys tied for the largest score in each row to be exactly the columns whose chart items are true. We show that bilinear scores of constant dimension cannot always represent this mask, even for charts generated by a fixed unambiguous grammar. Each head uses a common key bank across rows, with membership determined solely by its bilinear scores.

Proposition C.1 (Single-head limit for direct chart masks)**.** *There is a fixed unambiguous linear grammar such that, for every odd prime p, its exact chart on a suitable length-$(4p-1)$ word contains a $p \times p$ submatrix for which any bilinear, dimension-d head representing this submatrix as its rowwise top-tie sets requires*

$$d + 1 \;\ge\; \frac{p}{\sqrt{p} + 1}. \tag{34}$$

In particular, constant score dimension does not suffice.

Proof. Start from the fixed linear grammar $S \to 1 \mid aSb$, with $a, b \in \{0, 1\}$. Its strict-CNF presentation is

$$S \to 1 \mid TX, \qquad X \to ST, \qquad T \to 0 \mid 1.$$

Both presentations have the same S items and derive precisely the odd-length binary words with center symbol one: each recursion strips one terminal on each side, uniquely. All nonterminals are productive and reachable.

For an odd prime p, let χ be the quadratic character on $\mathbb{F}_p$: it is 1 on nonzero squares, -1 on nonsquares, and 0 at zero. Construct a word of length $4p-1$ by $w_{p+t} = \mathbf{1}[\chi(t \bmod p) \geq 0]$ for $0 \leq t \leq 2p-2$, and put zero elsewhere. For $0 \leq a, b < p$, choose starts $i_a = 2a$ and ends $j_b = 2p + 2b + 1$. The center is $p + a + b$. Hence the sign form of this exact S-chart minor is $H = C + P$, where $C_{ab} = \chi(a+b)$ and P is the permutation matrix selecting entries $a + b = 0$ in $\mathbb{F}_p$.

The character correlation is

$$\sum_{x \in \mathbb{F}_p} \chi(x+a)\chi(x+b) = \begin{cases} p-1, & a = b, \\ -1, & a \neq b. \end{cases} \tag{35}$$

For the off-diagonal case, shifting and scaling reduces the sum to $\sum_x \chi(x(x+1))$. The equation $y^2 = x(x+1)$ has $p-1$ solutions: $(2x+1-2y)(2x+1+2y) = 1$, parametrized by a nonzero first factor. Its solution count is also $p + \sum_x \chi(x(x+1))$, giving -1. Therefore $CC^{\mathsf{T}} = pI - J$, where J is the all-ones matrix. Thus $\|C\|_2 = \sqrt{p}$. Since $\|P\|_2 = 1$, $\|H\|_2 \leq \sqrt{p} + 1$.

The sign rank of H is the minimum rank of a real matrix with sign pattern H. Forster's theorem (Forster, 2002) gives $\operatorname{signrank}(H) \geq p/\|H\|_2$. Every row of H contains both positive and negative entries. If bilinear scores $\langle q_a, k_b \rangle$ realize exactly the positive entries as rowwise top ties, choose a threshold θ_a strictly between that maximum and the largest inactive score in row a. Writing the query and key matrices as Q and K, the matrix $QK^{\mathsf{T}} - \theta \mathbf{1}^{\mathsf{T}}$ has sign pattern H and rank at most $d+1$. Hence $d + 1 \geq p/(\sqrt{p}+1)$, even when global preprocessing chooses the embeddings. □

We next allow several heads and a Boolean combination of their top-tie membership bits. The following result uses the same grammar with a different family of input words.

Proposition C.2 (Multihead limit for direct chart masks)**.** *For the grammar above and every integer $h \geq 1$, there is a word of length $2^{h+1} + 2h - 1$ whose S-chart contains a $2^h \times h$ submatrix with the following property. Suppose each row is represented by a Boolean function of the top-tie membership bits of K bilinear heads of score dimension d. Each head uses keys shared across rows. The Boolean function may depend on the row but is the same for every column in that row. Then*

$$2^h \leq 2^{2^K} \Pi_d(h)^K, \qquad \Pi_d(h) = \sum_{r=0}^{\min(d+1,h)} \binom{h}{r}. \tag{36}$$

Consequently, fixed K and d cannot represent all such chart submatrices.

Proof. Let $P = 2^h$. A cyclic binary de Bruijn sequence of order h has length P and contains every h-bit word exactly once as a cyclic window (De Bruijn, 1946). Place this sequence, followed by its first $h-1$ bits, at word positions $P, \ldots, 2P + h - 2$, with zeros elsewhere. Use starts $i_a = 2a$, $0 \leq a < P$, and ends $j_b = 2P + 2b + 1$, $0 \leq b < h$, in a word of length $2P + 2h - 1$. Their centers are $P + a + b$. The selected chart rows therefore realize all 2^h patterns on the h selected columns.

For each head, fix its shared keys. A nonempty proper top-tie set is separated from the other keys by a threshold below the maximum score. Its restriction to the selected columns is therefore an affine halfspace trace; empty and full traces are also allowed. Affine halfspaces in $\mathbb{R}^d$ have VC dimension at most $d+1$ (Cover, 1965), so Sauer's bound (Sauer, 1972) gives at most $\Pi_d(h)$ such traces. Thus K heads yield at most $\Pi_d(h)^K$ tuples of traces. There are 2^{2^K} Boolean functions of K inputs, so allowing each row its own function gives at most $2^{2^K}\Pi_d(h)^K$ output patterns. Representing all 2^h chart rows therefore requires Eq. (36). For fixed K, d, its right-hand side grows polynomially in h, while the left-hand side grows exponentially. □

Explicit dependency relations can have cubic size even for unambiguous grammars. Direct chart masks, in turn, cannot always be represented by a fixed number of constant-dimensional bilinear heads with shared key banks and rowwise Boolean combinations of top-tie membership bits.

D EXPERIMENTAL PROTOCOLS AND SUPPLEMENTARY RESULTS

We detail the verified-update protocol, evaluation metrics, and experimental setup, then present eligibility and proposal-order comparisons, ranking diagnostics, and fixed-weight interventions. Waiting-time guarantees and height analysis connect scheduling to completion. A separate supplied-table study closes the appendix.

D.1 VERIFIED UPDATES AND EVALUATION

Verified updates. The chart starts from lexical items and their unit closure, and retains all verified items. The model proposes addresses $A[i, j)$ without supplying derivations. The verifier adds a proposed item when some rule $A \to BC$ and split k have both $B[i, k)$ and $C[k, j)$ in the chart before the update, recording the rule and split as its witness. Unit closure then adds all unit-rule consequences outside the proposal quota. The verifier computes all binary consequences before filtering by proposals and checks full chart closure for rejection. Write X_t for the chart after t updates.

Proposition D.1 (Sound stopping criteria)**.** *Under these verified updates, every item in X_t has a finite replayable derivation, regardless of the proposal sequence. The parser may stop with either of the following sound decisions:*

1. ***Acceptance.*** *If $S[0, n) \in X_t$, then $w \in L(G)$.*
2. ***Rejection.*** *If X_t contains all lexical bases, is closed under all unit and binary rules of G, and omits $S[0, n)$, then $w \notin L(G)$.*

Proof of Proposition D.1. We proceed by induction on the number of loops. Lexical matches give finite derivations for the initial items. Unit closure extends these derivations in dependency order, using the assumed acyclicity of unit rules. Each newly verified binary item receives a derivation by joining its two child derivations at the recorded split; both children belong to the chart before the update. The subsequent unit closure again attaches unit productions to already constructed derivations, including those added in this loop. Previously verified items retain their witnesses. Thus every retained item has a finite replayable derivation of its stated substring, and presence of the start item proves acceptance.

For rejection, let $U = X_t$ contain every lexical base and be closed under unit and binary rules. Then $I_0 \subseteq U$, and closure gives $I_r \subseteq U$ for every local round r by induction. Hence $I^\star \subseteq U$. Absence of $S[0, n)$ from U therefore proves rejection. □

Evaluation metrics. Sound-decision coverage in (6) counts inputs with a verified acceptance or rejection within the budget; other inputs remain unresolved. Positive-certificate rate is the fraction of positive inputs with a verified root within the same budget.

D.2 EXPERIMENTAL SETUP

Benchmarks and data. We use five fixed grammars: Dyck(1) and Dyck(2) with one and two bracket types, $a^n b^n$ with matched-pair recursion, Palindrome with matching outer terminals, and Arithmetic with operator precedence and parentheses. Grammars are binarized as needed, with acyclic unit rules. ID and OOD evaluation sets contain equal numbers of positive and negative inputs. Negatives are formed by length-preserving substitutions or adjacent swaps in positive words and verified as nonmembers by an exact parser. We select checkpoints on ID data, which may share words with training; OOD data use larger generation sizes and exclude same-seed training words.

Model architecture. The learned proposer uses token and positional embeddings to initialize its hidden states. Each loop applies the same transformer encoder block with softmax attention. Chart and proposal heads combine the two boundary states to score each nonterminal/span pair. Between loops, verified spans feed information between their endpoints, with feedback grouped by nonterminal. Auxiliary heads predict language membership and whether to halt.

Table 3: **Shared settings for the CFG scheduling experiments.** Dataset sizes are per grammar and seed; generation sizes specify the grammar-specific generator parameter. Within each of three paired seeds, A- and N-trained models share datasets, initial parameters, and batch order. The quota is computed from all m legal typed spans and remains unchanged when verified items are excluded.

Component	Setting	Value
Data	Dataset sizes	2,000 training; 240 ID; 240 OOD
	Generation-size parameter	1–6 for training and ID; 12–60 for OOD
Model	Hidden dimension	128
	Attention heads	4
	Feed-forward dimension	256
	Dropout	0.1
	Training loops	24, with shared parameters
Optimization	Optimizer	AdamW
	Learning rate	3×10^{-4}
	Weight decay	0.01
	Training epochs	30
	Training batch size	32
Evaluation	Loop budget	120; 240 for Arithmetic
	Proposal quota	$q = \lfloor 0.3m \rfloor$ per loop
	Checkpoint selection	Lowest ID task loss under N inference

Training and supervision. Models are trained separately for each grammar and policy. Exact charts supervise the chart head. At each loop, proposal targets are the missing binary items enabled by the current verified chart. The membership and halt heads receive auxiliary supervision; gradients do not pass through the verifier. Both policies use the same architecture and training settings (Table 3). All 24 loops execute from the start; supervision gradually extends across them.

Inference and checkpoint selection. In Table 3, $m = |\mathcal{N}|n(n+1)/2$ counts all legal typed spans. Both policies select up to the quota by proposal score: A allows all addresses, while N excludes verified items. At neural inference, selection uses no oracle targets. For each training policy, we select the checkpoint minimizing the sum of chart, membership, and halt losses on ID data under N inference. Each selected checkpoint is then evaluated with fixed weights under both A and N inference on identical ID and OOD inputs, using the budgets in Table 3. Coverage uses verified stopping criteria, independently of auxiliary predictions.

D.3 Eligibility and proposal-order comparisons

Paired estimation and uncertainty. For the crossed eligibility experiment (Section 4.2), let $Y_{g,s,t,e}$ denote final-budget OOD coverage for grammar g, paired seed $s \in \{1,2,3\}$, training policy t, and inference policy e. The primary contrast $NN - AN$ averages paired differences over the five grammars within each seed, then over seeds:

$$\delta_s = \frac{1}{5}\sum_{g}\big(Y_{g,s,N,N} - Y_{g,s,A,N}\big), \qquad \Delta = \frac{1}{3}\sum_{s=1}^{3}\delta_s. \tag{37}$$

The 95% confidence interval is $\Delta \pm 4.303\ \mathrm{sd}(\delta_1,\delta_2,\delta_3)/\sqrt{3}$, using a Student-$t$ approximation with two degrees of freedom. Secondary contrasts and interaction in Table 2(a) use the same aggregation and unadjusted intervals. Order and per-grammar comparisons are exploratory.

Proposal-order controls. Designed after the crossed experiment, the four controls retain its OOD cohorts and paired seeds under N eligibility, with the same verifier and budgets (Section D.2 and Table 3). They omit neural forward passes, so matching rounds and quotas does not match FLOPs or elapsed time.

Static selects the first q unverified items from a fixed address list at every loop. Rotating selects from the same list in cyclic order and advances its starting position by q in the original list after each loop. Random draws a fresh uniform permutation at every loop and selects the first q unverified items. Its

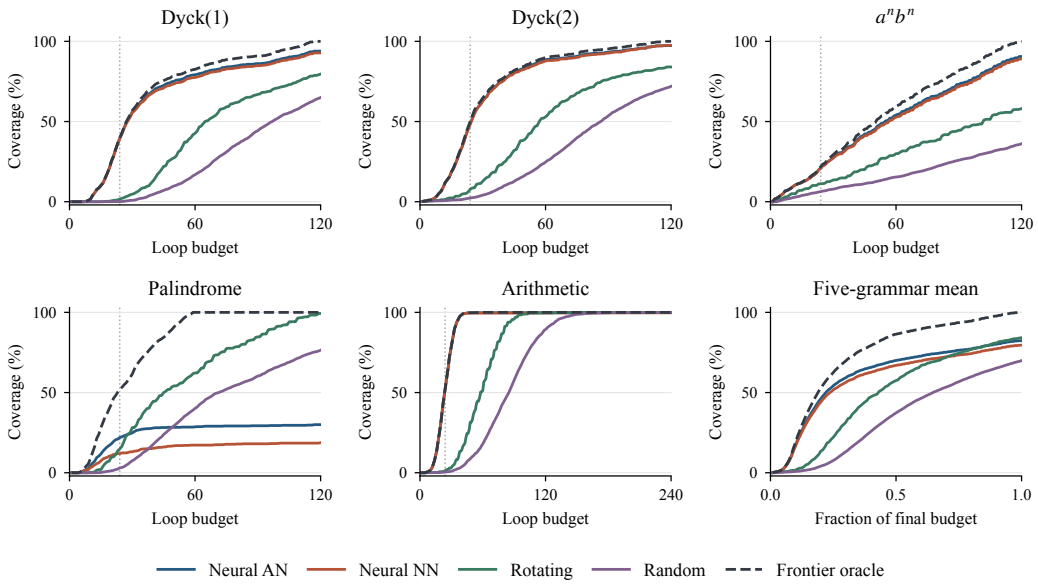


Figure 2: **Sound-decision coverage across loop budgets.** Curves show three-seed OOD means. The dotted line marks the 24-loop training horizon. The last panel averages grammars at equal fractions of their final budgets; Arithmetic uses twice as many loops. Random's three schedule repeats are averaged within each seed.

Table 4: **Completion on positive and negative OOD inputs (%).** Three-seed means under N inference at the budgets in Table 3. Pos. denotes root certification on positive inputs; Neg. denotes chart closure with no root on negative inputs. Random averages three repeats per seed.

Grammar	Neural AN		Neural NN		Random		Rotating	
	Pos.	Neg.	Pos.	Neg.	Pos.	Neg.	Pos.	Neg.
Dyck(1)	99.17	88.61	97.50	88.06	62.04	67.96	75.83	83.33
Dyck(2)	98.06	96.67	99.17	95.83	61.94	81.94	74.72	93.33
$a^n b^n$	100.00	81.11	100.00	78.06	14.72	57.59	36.67	79.44
Palindrome	34.44	25.56	21.39	16.11	60.00	92.87	98.61	100.00
Arithmetic	99.72	100.00	99.72	99.72	100.00	100.00	100.00	100.00

three schedule repeats per input and seed are averaged within each seed before seed aggregation; they do not add independent training runs.

The frontier oracle ranks enabled missing binary items first, using a fixed address order for ties, and fills any remaining quota from other unverified items. It serves as an optimistic reference using verifier information unavailable to the neural proposer.

Budget curves and the oracle reference. Figure 2 extends the final-budget comparison in Table 2(b). The oracle reaches 100% OOD coverage for all five grammars. Its entire enabled frontier fits within q at every update (Table 5), so each step equals exhaustive local propagation. By monotonicity, the oracle attains the fewest rounds to root discovery or chart closure on these inputs under the synchronous verified-update protocol. Since the verifier already computes the frontier, this comparison assesses proposal ordering, without establishing a need for learned rankings or a reduction in verification work.

Class-specific completion. Table 4 separates completion on positive and negative inputs. Rotating achieves nearly complete coverage in both classes on Palindrome. On $a^n b^n$, both neural policies certify all positive inputs, accounting for most of their advantage over Rotating; negative completion rates are similar.

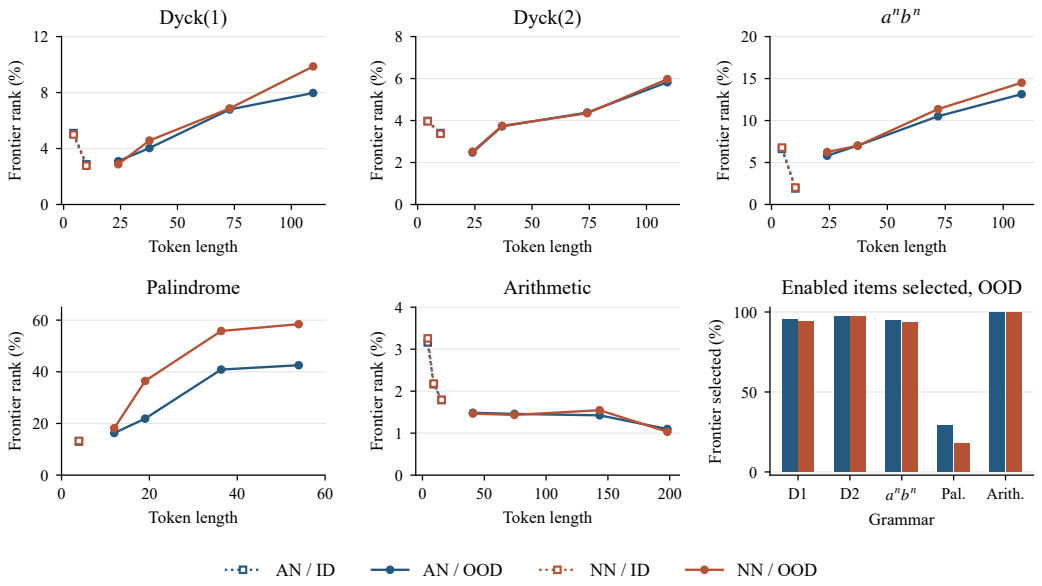


Figure 3: **Enabled-frontier ranks and selection under neural proposals.** The first five panels show the averaged median normalized rank (lower is better), at the mean token length of each bin. Vertical scales differ across these panels. Hollow squares are ID references with possible training-word overlap; filled circles are OOD. The last panel shows the mean selected fraction of the OOD frontier. Both training policies use N inference.

D.4 Ranking and progress diagnostics

Ranking diagnostics. We replay saved checkpoints for both training policies under N inference and record ranks without altering proposal selection. An enabled item's normalized rank is its position in the proposal order divided by the number of eligible candidates; lower is better. Frontier selection rate is the fraction of enabled missing items selected at an update. We include only unresolved updates with a nonempty frontier. For each input, we average the frontier's median normalized rank and selection rate over these updates, then average inputs within length bins.

Figure 3 shows how ranking varies with input length. For NN on Palindrome, the rank statistic worsens from 13.3% on ID inputs to 55.8% on 25–48-token OOD inputs. The selection rates in Section 4.3 show that few enabled items are selected on these longer inputs. The other grammars have smaller changes in normalized rank; Arithmetic's ranks remain low across input lengths.

Starvation diagnostics. A final plateau has no chart additions despite a nonempty frontier throughout the last 20 loops. On Palindrome, this occurs in about 93% and 90% of unfinished trajectories under A and N training, respectively, pooled across three seeds. Among unfinished trajectories at loop 120, the oldest enabled item's waiting time has a median of 118 loops under either policy. Spans of at most eight tokens account for about 85% and 80% of remaining enabled items, pooled over unfinished A- and N-trained trajectories, respectively. Most omissions concern short spans with verified children.

D.5 Fixed-weight scheduling interventions

For each training policy (A or N), we compare Neural, Alternating, and Switch after 24 using the same three checkpoints and Palindrome OOD inputs (Section 4.4). All schedules use N eligibility, the same proposal quota, and 120 loops, with model weights fixed. The network runs at every loop and receives the current verified chart, including updates made by Rotating, so subsequent neural proposals reflect those updates. Rotating's cursor starts at zero and advances by q in the full legal-address order only when Rotating supplies the proposals.

The coverage gains shown in Fig. 1, right, hold in every paired run under both training policies. On the evaluated inputs, neither intervention loses a decision completed by the neural baseline. Among inputs still unresolved at loop 120, neither intervention produces the final 20-loop plateau defined in Section D.4.

D.6 Waiting-time guarantees and height analysis

Completion depends on both derivation height and the waiting time for enabled items to be selected. Let H be the number of exhaustive local-update rounds needed for full chart closure. As in (9), it is the maximum of the minimum binary derivation heights of realizable items; unit closure is included in each round. Under exhaustive updates, positive inputs need only the root's minimum height $h_+ \le H$ for acceptance; negative inputs require $h_- = H$. Write h for this label-dependent decision height.

Bounded waiting and chart completion. For an order π on the fixed legal universe $\mathcal{I}_n$, let $Q_\pi(X)$ be the first $\min(q, |\mathcal{I}_n \setminus X|)$ elements of $\mathcal{I}_n \setminus X$ in order π. The verified update is

$$F_\pi(X) = \mathrm{unitcl}\big(X \cup (D(X) \cap Q_\pi(X))\big). \tag{38}$$

The quota q is fixed from the full legal universe throughout the run.

Proposition D.2 (Bounded waiting under cyclic updates)**.** *Use a fixed universe of m legal items, complete lexical initialization, and a quota $1 \le q \le m$. Every $p \ge 1$ loops, a cyclic update selects the first q unverified items in a fixed cyclic order (or all if fewer remain) and advances its starting position by q in the original m-item order. Retain all verified items, accept every selected enabled item, and allow only sound neural updates between cyclic steps. Apply unit closure at initialization and after each update. Then every continuously enabled missing item is added within $B = p\lceil m/q \rceil$ loops. If exhaustive local updates saturate the chart in H rounds, the combined process saturates within BH loops.*

Proof of Proposition D.2. First show that $X \subseteq Y$ implies $F_\pi(X) \subseteq F_\pi(Y)$. If $a \in D(X) \cap Q_\pi(X)$ is already in Y, it is retained. Otherwise, $a \in D(Y)$ by monotonicity of binary inference. Removing the additional known items in $Y \setminus X$ cannot increase its eligible rank, so $a \in Q_\pi(Y)$. Unit closure is monotone as well. This proves the update inclusion. Every intervening sound update contains its input chart. Hence, after any number of cyclic updates, the combined chart contains the chart obtained by those same cyclic updates alone from the same initialization.

Index cyclic updates from zero. Update s starts at rank $sq \bmod m$. Every still-missing item in the raw cyclic window of q consecutive addresses is among the first q eligible addresses. Any $b = \lceil m/q \rceil$ consecutive cyclic windows cover all m addresses, since their starts advance by exactly q. A continuously enabled missing item is therefore selected and accepted within b cyclic updates, or $B = pb$ total rounds.

Write I_0 for lexical and unit initialization and $I_{r+1} = \mathrm{unitcl}(I_r \cup D(I_r))$. Inductively, the combined chart contains I_r by round Br: once the children for an item in $D(I_r)$ are present, that item stays enabled until selected, which takes at most B further rounds. If $I_H = I^\star$, soundness and this inclusion imply equality by round BH. Moreover, $H \le n - 1$ by Proposition A.1. The same induction places a positive input's root in the chart by Bh_+ loops, while negative inputs require closure. Thus Bh also bounds the loops to a decision for either label. □

Inferring closure from stalled updates. A chart may stop growing while enabled items wait for selection. With a waiting bound B, a stall lasting B rounds certifies closure; unrestricted neural proposals need not satisfy this bound.

Claim D.3 (Bounded waiting and closure)**.** *Let chart retention be monotone. Suppose that, from the start of any round, every enabled missing item is proposed during that round or the next $B - 1$ rounds unless it has already been added, and the verifier accepts every proposed enabled item. Then B consecutive rounds without an addition imply closure. Together with complete lexical initialization and absence of the start item, this condition gives sound rejection. Requiring only that every continuously enabled missing item is eventually proposed allows arbitrarily long finite periods without an addition.*

Table 5: **OOD lengths, binary heights, and cyclic-schedule bounds.** Height entries are median / 95th percentile, pooled over three seeds with 360 positive and 360 negative input occurrences per grammar. The $bh \leq T$ column gives the percentage satisfying Rotating's sufficient decision bound, where $h = h_+$ on positive inputs and $h = h_-$ on negative inputs, with $b = 4$ and $T = 120$ (240 for Arithmetic). The last column gives the maximum frontier-to-quota ratio along oracle runs.

Grammar	Token length	h_+	h_-	$bh \leq T$ (%)	Max. $\lvert F \rvert / q$ (%)
Dyck(1)	24–120	30 / 105	26 / 102.1	56.9	2.7
Dyck(2)	24–120	29 / 101.1	21 / 57.1	64.7	1.7
$a^n b^n$	24–120	71 / 113	30.5 / 89	29.3	0.6
Palindrome	12–60	34 / 56	16 / 41	61.5	18.8
Arithmetic	31–209	28 / 38	21 / 32	100.0	0.8

Intervention budgets and measured heights. In the experiments, $b = \lceil m/q \rceil = 4$. Alternating uses a cyclic update every other loop, giving $B = 8$; Switch after 24 gives $B = 4$ after the switch, with 96 loops remaining. Sufficient conditions for a decision are therefore $8h \leq 120$ and $4h \leq 96$, using the decision height from initialization. The no-growth window in Claim D.3 also starts after the switch. These bounds charge the maximum wait at every level; actual waits may be shorter, and the switch bound does not credit progress made during the first 24 loops.

Table 5 reports input lengths, decision heights, and the fraction satisfying pure Rotating's decision bound $bh \leq T$. Every OOD decision height is within the budget, and every oracle frontier fits within quota, explaining the oracle's full coverage. The fraction with $h \leq t$ equals Oracle coverage at loop t. On Palindrome, the decision-height conditions for Alternating and Switch after 24 hold for only 28.1% and 51.9% of inputs, respectively, well below their observed coverage. The worst-case bounds are thus sufficient but conservative.

Coverage conditional on height. Figure 4 compares NN and Rotating on the same inputs, grouped by label and decision height relative to budget, h/T. For positive Dyck(1) inputs with $h/T > 0.5$, NN reaches 100% coverage versus 25.0% for Rotating. For positive Palindrome inputs with $0.25 < h/T \leq 0.5$, the ordering reverses: NN reaches 2.3% versus 97.7% for Rotating. Scheduling differences persist within height groups, although inputs may still differ in other grammatical structure.

Random waiting versus bounded waiting. Under Random, each loop samples q unverified items uniformly without replacement, or selects all if fewer remain. Set $\alpha = q/m$ with $0 < q < m$. A continuously enabled missing item is selected with conditional probability at least α, since at most m items are eligible. Its waiting time W, measured from its first enabled update, satisfies

$$\Pr(W > k \mid \text{history at first enablement}) \leq (1-\alpha)^k, \qquad \mathbb{E}[W] \leq 1/\alpha. \tag{39}$$

Enabledness persists as the chart grows. A union bound over at most m items gives, with probability at least $1 - \delta$, a simultaneous waiting bound

$$k_\delta = \left\lceil \frac{\log(m/\delta)}{-\log(1-\alpha)} \right\rceil, \qquad 0 < \delta < 1. \tag{40}$$

Induction on derivation height then bounds saturation by $k_\delta H$ loops on this event. Random thus has a probabilistic completion bound, while cyclic scheduling guarantees BH deterministically. The experiments do not isolate the contribution of random waiting-time tails from other ordering effects.

D.7 Rule reuse from supplied transition tables

Retrieval setup. This study tests execution of transition rules supplied in context, including rules for unseen automata. Each input contains a serialized deterministic visibly pushdown automaton (DVPA) (Alur & Madhusudan, 2004) and a word. The input symbol determines whether to push, pop, or leave the stack unchanged. A pointer queries the transition table by current state, input symbol, and stack top when required. For a complete deterministic table, positive equality weights make the matching row the unique maximizer. Supplied table parsing, value decoding, and state/stack updates turn each retrieval into an execution step. Thus exact execution combines the retrieval interface's matching guarantee with a supplied executor.

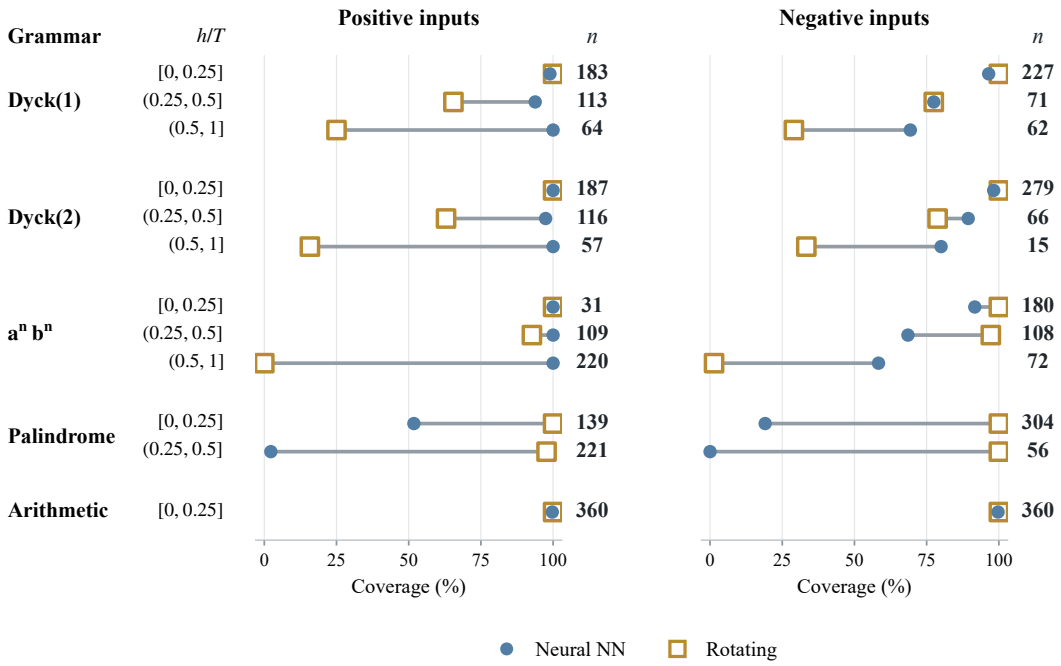


Figure 4: **OOD coverage by decision height relative to budget. Left:** positive inputs, with $h = h_+$. **Right:** negative inputs, with $h = h_- = H$. Within each height group, circles and squares compare NN and Rotating on identical inputs; segments connect their coverage values. The n columns give group counts, pooled over three seeds with 360 positive and 360 negative input occurrences per grammar. All nonempty groups are shown.

Encoder comparison and evaluation. The first comparison evaluates a token encoder against a pointer using fixed equality scores and one trainable positive scale, which preserves row rankings. The models are compared across three paired seeds with matched training exposure; parameter counts and computational costs differ. Evaluation covers four tiers with balanced labels: training examples, new words for seen automata, and unseen automata at ID and OOD lengths. The new-word tier excludes training words. Membership uses balanced accuracy over words. Transition-step accuracy requires the correct next state, action, and stack top at a token; exact-trace-token accuracy additionally requires the correct previous state and prefix-acceptance status. Both metrics are reported as fractions of tokens. The pointer scores 1.0 on all three metrics in every evaluated tier. On unseen automata at OOD lengths, the encoder's corresponding means are 0.5000, 0.0945, and 0.0249.

Retrieval interfaces and controls. The second comparison trains equality-coordinate weights and a score scale for two interfaces: materialized rule rows and raw token windows. Each arm has 15 trainable parameters and uses the same 45-epoch exposure across three paired seeds. For raw windows, a supplied local scanner identifies rule boundaries and validates complete, unique keys; attention retrieves the corresponding windows from the serialized input. Both interfaces retain supplied decoding and execution and score 1.0 on all three metrics in every tier. The raw-window model's final equality weights are positive in all three runs, satisfying the matching condition above. Two controls test whether execution uses input rules: permuting rule rows leaves execution unchanged, whereas modifying serialized transition values while keeping the original automaton object fixed changes execution and can flip membership. These controls confirm rule reuse.

E BROADER IMPACT AND FUTURE WORK

E.1 BROADER IMPACT

Replayable certificates make accepted parses checkable against the grammar. Distinguishing verified decisions from unresolved inputs clarifies finite-budget completion. Separating proposals from verification lets learned components change while correctness checks remain fixed, supporting comparisons of learned and deterministic schedules under a common protocol. Such interfaces may support other symbolic tasks with explicit rules and checkable outputs.

E.2 LIMITATIONS AND FUTURE WORK

Our constructions assume fixed grammars and averaging-hard attention; the experiments study scheduling with an external verifier. Future work could extend blank initialization to the UCFL storage bound $\mathcal{O}(n^{\omega_{\mathbb{Q}}+\varepsilon})$ and reduce initialization depth for general CFLs. More compact context representations and implementations with standard soft attention would broaden the applicability of the constructions. On the learning side, studies across more grammars, input lengths, and training scales could clarify when learned scheduling improves completion. Learning more of the parsing pipeline while retaining checkable outputs, and measuring end-to-end computation including verification, would connect these results to practical parsing efficiency.