CANOPY: CLOSED-FORM NEUROSYMBOLIC COMPOSITION WITH A SMALL STATE-SPACE REDUCER

Anonymous authors
Paper under double-blind review

ABSTRACT

Compositional reasoning benchmarks such as long-form LRA-ListOps and the DeepMind Mathematics arithmetic suite are tree-evaluation problems written as linear token streams. A model reading only the tokens must recover the tree structure, a language problem, and execute the operator rule at each node, a symbolic one. These carry different inductive biases, and end-to-end sequence models continue to trail systems that route inputs through an explicit composition rule.

CANOPY separates the two at inference time. A parser supplies the tree, a small state-space reducer approximates each atomic operator call as a probability-mass function (PMF) over the answer, and a fixed closed-form composer propagates those distributions up the tree. Reducer and composer exchange nothing but PMFs, so either can be swapped or retrained without touching the other. A sub-1M-parameter reducer reaches $\mathbf{0.9535}$ on long-form LRA-ListOps, above the tree-induction band and above end-to-end baselines carrying up to $90\times$ more parameters. On DM Math addition chains it reaches $\mathbf{0.989}$ against a 30M-parameter reference at ~ 0.94.

Mambino, the reducer instantiated here, carries an internal predictor branch that feeds its own one-step prediction error back into the recurrence. We partition every evaluation by whether an operator call was seen in training. Ablating the branch costs 2.8 pp or less on seen calls, and 25.0, 11.3 and 5.1 pp on unseen multiplication, addition chains and single-operation addition, respectively. On LRA-ListOps, whose operators are saturating, ablating it instead improves composition by 4.45 pp. The branch earns its parameters only where the reducer must compute rather than recall.

1 INTRODUCTION

Compositional reasoning benchmarks (Long Range Arena ListOps (Tay et al., 2021) and the DeepMind Mathematics arithmetic suite (Saxton et al., 2019)) are tree-evaluation problems written as linear token streams. A network that sees only the token sequence must jointly recover the tree structure (a language sub-task) and execute the operator rules (a symbolic sub-task), two problems with different inductive biases bundled into one learning objective. Selective-scan SSMs (Soydan et al., 2024) and attention–SSM hybrids (Ma et al., 2023) climb into the low sixties on LRA-ListOps, yet remain well below the tree-composer band on the same split.

We present **CANOPY**, a parse-then-compose framework in which a supplied parser recovers the operator tree, a reducer emits a PMF per atomic operator call, and a fixed closed-form composer propagates those distributions upward. **Mambino** is the reducer architecture CANOPY instantiates here: the predictor-branch augmentation of a diagonal-scan SSM introduced in §3, trained on isolated operator calls and invoked at atomic-reduction nodes only at test time. A single reducer handles both problem regimes: aggregation over saturating operators like MAX and majority, where it reaches $\mathbf{0.9535}$ on long-form LRA-ListOps; and carry-propagating multi-digit arithmetic, where it reaches $\mathbf{0.989}$ on `arithmetic.add_sub_multiple`, beating the 30M Saxton Transformer by $\mathbf{4.9}$ pp. That comparison is scoped to the one DM Math module whose evaluated population matches that of Saxton et al. (2019): integer operands under a magnitude clip. The framing is neurosymbolic in the sense of the Neuro-Symbolic Concept Learner (Mao et al., 2019) and the wider parser-equipped-composer lineage (Tai et al., 2015): symbolic infrastructure carries the compositional structure; the neural component supplies the primitive operation.

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.

Attached to the base state-space block is an *internal-state predictor branch*, a small head that forecasts the reducer's own next input and injects the resulting error back into the recurrence. Its contribution is measured against training overlap, not in aggregate. When every DM Math evaluation is partitioned by whether the operator call was seen in training, ablating the branch costs 2.8 pp or less on seen calls and roughly 25 pp on unseen multiplication. The effect is graded by how much positional carry an operator demands (§6.2). On LRA-ListOps' saturating operators the opposite holds and removing the branch improves composition by 4.45 pp. The pattern is consistent with the constant-depth TC^0 ceiling of diagonal-scan SSMs identified by Merrill et al. (2024): the branch earns its parameters only where the reducer has to compute a result rather than recall one.

A second Mambino variant, **Mambino-Residual**, targets multi-digit multiplication chains. It is a residual μ-refiner over a symbolic naive-product baseline, engaged only where the composer's place-value rule leaves residual complexity for the reducer to model (§5.2).

The distinct claim, situated against the parser-plus-composer lineage surveyed in §2, is that once the composer is fixed as a closed-form rule table over its input PMFs, reducer and composer become byte-independent modules. Either can be swapped or retrained without touching the other, which §6 validates across reducer architectures.

We extend the closed-form PMF composer to cover the four LRA-ListOps operators and the ADD/SUB/MUL DM Math family, bit-exact on one-hot reducer outputs ($5000/5000$; DIV takes a numeric shortcut with bounded truncation). We give two composition algorithms - hard-argmax recursion (§4.1) and closed-form PMF propagation (§4.2) - that both treat the reducer as a byte-independent module. The ablations of §6 then characterize the predictor branch as a task-conditional piece, load-bearing on carry-propagating operators applied to unseen inputs and redundant elsewhere.

2 Related Work

Architectural background. Our reducer inherits the S5 diagonal scan (Smith et al., 2023), a simplification of S4 (Gu et al., 2022). It does not use the selective-scan variants (Mamba (Gu & Dao, 2023), S7 (Soydan et al., 2024), Mamba-2 (Dao & Gu, 2024)), whose dynamic per-timestep parameters target long end-to-end sequences that the reducer never sees. The predictor branch mirrors the forward-model / residual loop of predictive coding and active inference (Rao & Ballard, 1999; Friston, 2010; Feldman & Friston, 2010; Lotter et al., 2017; Millidge et al., 2022), realized as a second diagonal-scan SSM.

Benchmarks and probabilistic-neurosymbolic composers. LRA-ListOps (Tay et al., 2021) is a four-operator (`MIN`/`MAX`/`MED`/`SM` mod 10) tree-arithmetic task at up to 2048 tokens; recent locality-bias analysis (Miralles-González et al., 2025) shows much of end-to-end performance on it is short-range recoverable. DM Math (Saxton et al., 2019) covers 56 modules with a 30M reference Transformer at $\sim 76\%$ aggregate. The probabilistic-neurosymbolic composer lineage (Manhaeve et al., 2018; Li et al., 2023; Ahmed et al., 2022; Mao et al., 2019; De Smet & Zuidberg Dos Martires, 2024) composes at the PMF layer but couples the composer to the neural predicate at training time. Ours is a fixed closed-form tensor recursion at inference time, extending the closed-form family to order statistics and multi-digit RV $\times$ RV multiplication.

Tree-structured composers and SSM expressivity. Tree-LSTMs and their extensions (Tai et al., 2015; Maillard et al., 2017; Bowman et al., 2016; Kim et al., 2019; Nye et al., 2020) entangle composer and leaf in one learned recursive cell. On the long-form 2K-token ListOps split the strongest published numbers are 0.801 for Ordered Memory (Shen et al., 2019), 0.718 for BT-GRC+OneSoft (Chowdhury & Caragea, 2023), and 0.554 for CRvNN (Chowdhury & Caragea, 2021) (Ordered Memory and CRvNN re-measured by Chowdhury & Caragea 2023). All three induce the tree inside the learned cell at inference time. We recover the tree with a fixed external parser and compose the reducer's PMFs with a closed-form rule table. Neural Arithmetic Logic Units (Trask et al., 2018; Madsen & Johansen, 2020) add differentiable arithmetic primitives, but are arithmetic-specific and do not extend to LRA-ListOps' order-statistic and modular-sum operators. Merrill et al. (2024) place diagonal-scan state-space models of the S4/Mamba family inside TC^0, ruling out per-token state whose complexity grows with input length. Our §6.3 operator-regime split maps onto that picture.

3 Architecture

Mambino names the predictor-branch augmentation of a diagonal-scan SSM; this paper uses a Mambino block as the atomic operator reducer inside CANOPY. The reducer is a stack of real-diagonal

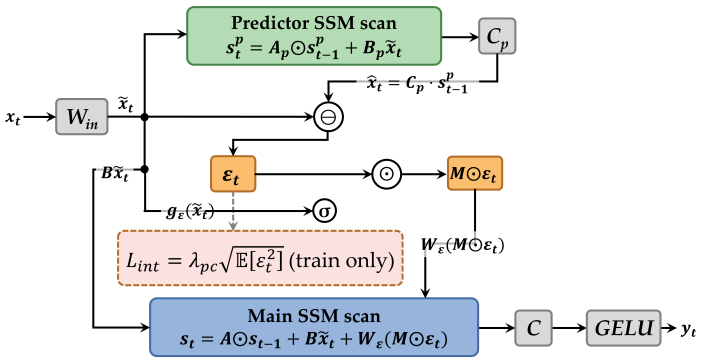


Figure 1: Mambino reducer architecture: two parallel real-diagonal SSM scans over the projected input $\tilde{x}_t$. The **Predictor** branch (green, top) emits a causal one-step-ahead prediction $\hat{x}_t$; the **Main** SSM (blue, bottom) integrates the projected input and the sigmoid-gated surprise $W_\varepsilon(M_t \odot \varepsilon_t)$, $\varepsilon_t = \tilde{x}_t - \hat{x}_t$. Equations, mask M, and regularizer: §3.2.

state-space blocks with one non-standard addition, an internal-state predictor branch (Figure 1, Section 3.2) that injects a continuous prediction-error signal into the residual stream. The base recurrence $h_t = Ah_{t-1} + Bx_t$, $y_t = Ch_t$ follows the real-diagonal simplification of S5 (Smith et al., 2023), and a sigmoid mask M gates the predictor branch's error into the recurrence. Training is supervised cross-entropy on the operator's output token, averaged over isolated operator calls; full definitions, eigenvalue parameterization, and hyperparameters are in Appendix A.

3.1 Base state-space block

Each layer consumes a token stream $x_{1:T}$ at hidden dimension d, projects to a state of dimension d_C via $\tilde{x}_t = W_{\text{in}}\, x_t + b_{\text{in}}$, and runs a real diagonal state-space recurrence

$$s_t = A \odot s_{t-1} + B\, \tilde{x}_t, \tag{1}$$

where $A \in \mathbb{R}^{d_C}$ has each element learned in $[\lambda_{\min}, \lambda_{\max}] \subset (0, 1)$ via a sigmoid parameterization (Appendix G); every trained eigenvalue in the released checkpoints lies in $(0, 0.97]$, while each dimension finds its own timescale. The readout is

$$y_t = \text{GELU}\big(C \cdot s_t\big). \tag{2}$$

The input, ε-injection, and output paths each carry a learned sigmoid gate. Real-diagonal A (no complex-conjugate pairs) and the dropped direct-feedthrough term D from the classical LTI readout together identify this block as the real-diagonal simplification of S4 (Gu et al., 2022) introduced by S5 (Smith et al., 2023). The parallel FFT-based scan gives $O(T \log T)$ time and $O(T\, d_C)$ memory per layer, with per-reducer constants (d, d_C, L, sequence lengths) listed in Appendix G.

3.2 Internal-state predictor branch

Each layer carries a second real-diagonal SSM of identical form alongside the main recurrence. It plays the role of an *efference-copy / forward-model* pathway of the kind proposed for cortical predictive coding and cerebellar motor control (Rao & Ballard, 1999; Wolpert & Ghahramani, 2000; Friston, 2010). The predictor scan emits its own next-input estimate $\hat{x}_t$, and the residual $\varepsilon_t = \tilde{x}_t - \hat{x}_t$ (the *surprise*) is fed back into the main state via a learned gain W_ε, correcting downstream dynamics in proportion to input surprise rather than input magnitude:

$$s_t^p = A_p \odot s_{t-1}^p + B_p\, \tilde{x}_t, \quad \hat{x}_t = C_p \cdot s_{t-1}^p, \tag{3}$$

$$s_t = A \odot s_{t-1} + B\, \tilde{x}_t + W_\varepsilon \cdot \big(M_t \odot \varepsilon_t\big), \qquad M_t \;\equiv\; \sigma\big(g_\varepsilon(\tilde{x}_t)\big), \tag{4}$$

A_p is parameterized identically to A, and M_t is the entry-wise sigmoid gate on surprise labeled "M" in Figure 1. An auxiliary loss $\lambda_{\text{pc}} \cdot \sqrt{\mathbb{E}[\varepsilon_t^2]}$ with $\lambda_{\text{pc}} = 0.1$ regularizes the predictor toward low-error forecasts, but the load-bearing effect is not tight forecasting; it is that ε_t enters the main scan's state via W_ε.

4 Composition Algorithms

Given a supplied operator tree, two inference-time composition algorithms drive the reducer.

4.1 ALGORITHM 1: HARD-ARGMAX RECURSION

The first algorithm collapses each child distribution to its most-probable integer and re-invokes the trained reducer at every internal node. It requires only an argmax-recoverable answer at each call and discards the reducer's full output distribution at every recursion step.

Algorithm 1 Hard-argmax recursion

procedure EVAL(node)
  if node is a literal operand **then**
    return integer value of node
  else
    op $\leftarrow$ operator symbol at node
    $c_1, \ldots, c_k \leftarrow$ children of node
    $v_i \leftarrow \text{EVAL}(c_i)$ for $i = 1, \ldots, k$
    $p \leftarrow \text{LEAF}(\text{op}, v_1, \ldots, v_k)$ ▷ leaf emits softmax over integer output
    return $\arg\max_y p[y]$
  end if
end procedure

Algorithm 1 is architecturally agnostic, but argmax collapse discards the reducer's uncertainty and the loss compounds through the tree; under naive multiplicative independence a reducer with 97% top-1 per call is bounded by 0.97^m end-to-end over m leaf calls per tree (§5.1).

4.2 ALGORITHM 2: CLOSED-FORM PROBABILITY-MASS-FUNCTION PROPAGATION

The second algorithm avoids argmax collapse. The reducer is invoked only at *atomic-reduction* nodes (internal nodes whose children are all literal operands); every strictly-internal node reconstructs its answer PMF from its children's PMFs via a closed-form operator rule, walked deepest-first (pseudocode below).

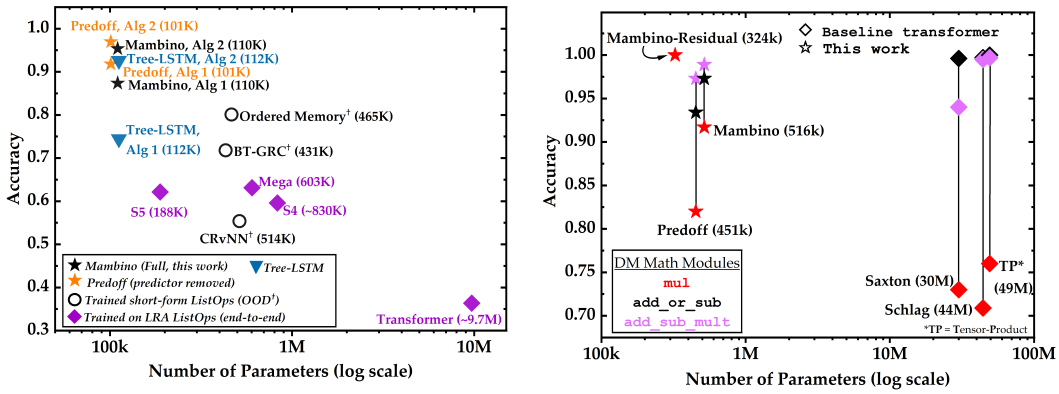
Algorithm 2 Closed-form probability-mass-function propagation

procedure EVALPMF(node)
  if node is a literal operand **then**
    return δ_v where v is the integer value of node ▷ one-hot PMF
  else if every child of node is a literal operand **then**
    op $\leftarrow$ operator symbol at node
    $v_1, \ldots, v_k \leftarrow$ literal values of children
    return $\text{LEAF}(\text{op}, v_1, \ldots, v_k)$ ▷ leaf softmax over integer output; one neural call per atomic-reduction node
  else
    op $\leftarrow$ operator symbol at node
    $c_1, \ldots, c_k \leftarrow$ children of node
    $p_i \leftarrow \text{EVALPMF}(c_i)$ for $i = 1, \ldots, k$ ▷ child PMFs
    return $\Phi_{\text{op}}(p_1, \ldots, p_k)$ ▷ closed-form rule for parent operator
  end if
end procedure
final decode: $\hat{y} \leftarrow \arg\max_y \text{EVALPMF}(\text{root})[y]$

The operator-specific rules Φ_{op} are closed form. For LRA-ListOps, `MIN` propagates as a product of survival functions, `MAX` as a product of CDFs, `SM` mod 10 as a mod-10 circular convolution over the residue ring, and `MED` by joint enumeration over the child support (full statements in Appendix B). For DM Math arithmetic, Algorithm 2 uses per-digit convolutional and partial-product rules plus a mixed-sign case-split (Section 4.3, Appendix D). Under calibrated softmax inputs the propagation carries the reducer's uncertainty upward, subject to independence assumptions between digit positions and between sign–magnitude marginals that introduce a small controlled approximation. Carrying uncertainty this way recovers accuracy above Algorithm 1's hard-collapse floor.

4.3 DM MATH REPRESENTATION AND REDUCER VARIANTS

DM Math operands and answers are $K = 18$-digit signed decimals (12 integer + 6 decimal positions, one 3-way sign). Two reducer variants share the SSM trunk of §3 but differ in input, readout, and training objective. **Mambino**, a position-wise categorical reducer, composes under both Algorithms 1



(a) Long-form LRA-ListOps (2048-token). (b) DeepMind Mathematics arithmetic (per module).

Figure 2: **Accuracy versus parameter count**, log-x. Panel (a) shows the LRA-ListOps Mambino, panel (b) the DM Math Mambino and Mambino-Residual (the latter on `mul` only). Marker **shape**: ★ this work (parser-supplied SSM reducer); ▼ this work, Tree-LSTM reducer (§6.3, panel a only); ♦ end-to-end SSM/Transformer; ◯ tree-induction RvNN (panel a only). In panel (b) **color** encodes the DM Math module (in-panel legend), with a vertical line joining each system's points. "Full" = predictor branch active; "Predoff" = branch removed (§6). Values for this work in Tables 1 and 2; baseline points are taken from the cited papers.

and 2. **Mambino-Residual**, a MUL-specific residual μ-refiner over a symbolic-naive-product baseline, composes under Alg. 1 only. Full definitions, equations, and the Alg. 2 propagation rules for Mambino are in Appendix C.

Integer-operand invariant. Under Alg. 1 every internal composer node returns an integer. This is a design property enforced at four independent points of the DM Math driver: the sample sanitizer `_ast_to_expr` rejects non-integer constants at parse time, the ground-truth evaluator `_eval_safe` cannot construct a non-integer answer, the operand tokenizer `parse_expr` accepts integer literals only, and the reducer's re-tokenization step applies `int(round(·))` before feeding a child's return value to a parent. The evaluated population is therefore integer operands under $|\text{operand}| \leq 999$ and $|\text{result}| \leq 999$; the non-integer part of DM Math's arithmetic modules is out of scope for this pipeline.

5 Results

We report per-configuration exact-match accuracy for the two composition algorithms across LRA-ListOps and the DM Math arithmetic modules. Every numerical claim is anchored to a released checkpoint alias and evaluation-artifact alias in the reproducibility catalog (Appendix L).

Figure 2 previews the parameter-efficiency frontier: on both benchmarks our sub-1M-parameter parser-supplied reducers sit above published end-to-end systems one to two orders of magnitude larger.

5.1 LRA-ListOps

Mambino, the LRA-ListOps atomic operator reducer, has 110,192 parameters and is trained on individual operator calls (`MIN`, `MAX`, `MED`, `SM`) at literal-operand tuples ($N \in \{2..10\}$ arguments). On the 2,000-expression test partition, the same reducer reaches:

- **0.8735** test accuracy under Algorithm 1 (hard-argmax recursion), and
- **0.9535** test accuracy under Algorithm 2 (closed-form PMF propagation via Equations 7–10 of Appendix B).

Table 1 compares our numbers to prior tree-composition and end-to-end sequence-model results on the long-form LRA-ListOps split.

The 8.0-percentage-point gap between the two algorithms on the same reducer and test split is the empirical signature of argmax collapse: Alg. 1 discards the reducer's uncertainty at every internal node, while Alg. 2 carries it upward through the exact operator rule. Figure 3 illustrates the mechanism on a minimal example.

Table 1: LRA-ListOps (long-form, 2048-token) accuracy. ‡Re-measured on this split by Chowdhury & Caragea (2023) (Table 1); the originals (Shen et al., 2019; Chowdhury & Caragea, 2021) report short-form only. §Sonnet 4.5 protocol (shared by Table 2): model `claude-sonnet-4-5-20250929` via the Anthropic Messages API, zero-shot, one call per problem, no scratchpad, no tool use. DM Math samples drawn through the release's `experiments.dm_math_extract.extract_pair` filter (matches the DM Math Mambino's evaluated population; integer operands with $|\cdot| \leq 999$, $n = 200$ per module). LRA-ListOps samples are the first $n = 100$ indices of the long-form 2K-token test partition (`fengyang0317/listops-1000`, matches the LRA-ListOps Mambino's evaluated population). Responses that do not parse to the required answer format (single 0–9 digit for LRA, numeric for DM Math) are scored incorrect; $23/100$ LRA responses fall into this category. Per-sample transcripts released as artifacts `eval-A24z–eval-A30z`. †Cross-architecture ablation (§6.3): SSM reducer swapped for an N-ary Tree-LSTM cell (Tai et al., 2015) at matched budget (112K vs. 110K), same parser, composer, and curriculum.

System	Class	Test acc.
Ordered Memory (Shen et al., 2019)‡	tree-induction cell (structure learned)	0.801
BT-GRC+OneSoft (Chowdhury & Caragea, 2023)	tree-induction cell (structure learned)	0.718
CRvNN (Chowdhury & Caragea, 2021)‡	tree-induction cell (structure learned)	0.554
S7 (Soydan et al., 2024)	selective SSM (e2e)	$\sim$0.638
Mega (Ma et al., 2023)	attn–SSM hybrid (e2e)	$\sim$0.631
S5 (Smith et al., 2023)	diagonal SSM (e2e)	$\sim$0.622
Claude Sonnet 4.5, zero-shot§	LLM	0.120
Mambino, Alg. 1 (this work)	parser + SSM reducer	**0.8735**
Mambino, Alg. 2 (this work)	parser + closed-form PMF	**0.9535**
Tree-LSTM reducer†, Alg. 1 (this work)	parser + Tree-LSTM reducer	0.7445
Tree-LSTM reducer†, Alg. 2 (this work)	parser + closed-form PMF	**0.9260**

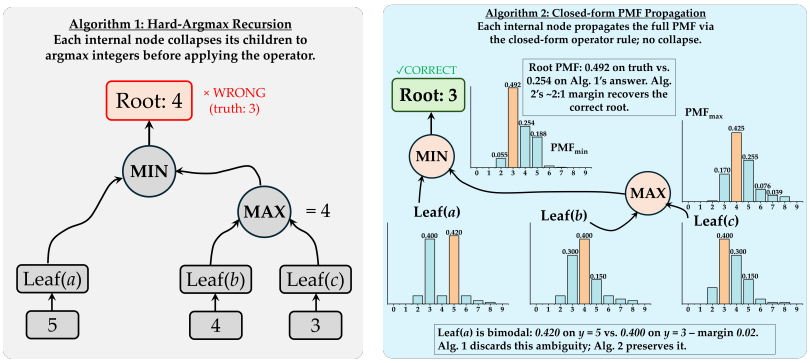


Figure 3: Same trained Mambino reducer on $\texttt{MIN}(a, \texttt{MAX}(b, c))$, root $= 3$, inference only. **Alg. 1** (left): two leaf misclassifications compound to the wrong root. **Alg. 2** (right) carries PMFs upward and recovers the correct root at $\sim$2:1.

What the parser supplies. LRA-ListOps expressions are fully parenthesized: the parser is bracket matching ($\sim$80 LOC). The DM Math parser absorbs a full linguistic wrapper across 56 modules and supplies typed operands ($\sim$600 LOC). This parsing code is uncounted in every parameter comparison in Table 1. The tree-induction rows learn compositional structure jointly with the primitive; end-to-end models learn it implicitly; both pay the full budget. When compositional structure is analytically available, inducing it is not worth the parameter cost, and the residual primitive is small.

The LRA test set averages $\sim$147 leaf calls per tree (294,843 nodes across 2000 trees). Under naive multiplicative independence a 97%-per-call reducer would compound to only $0.97^{147} \approx 1\%$ joint accuracy (a worst-case independence bound); the observed 87.35% under Alg. 1 is a consequence of *error masking*. For set-reducing operators like `MAX`, `MED`, and `MIN`, a wrong non-extremal child leaves the operator's output unchanged, so most sub-argmax errors do not propagate. Alg. 2 adds $+8.0$ pp by carrying the reducer's uncertainty upward instead of argmax-collapsing it at every node. A root-operator label-prior baseline is supplied in Appendix H.

Zero-shot LLM baseline. As a reference point outside the tree-composer and end-to-end SSM bands, Claude Sonnet 4.5 zero-shot on $n = 100$ long-form LRA-ListOps expressions from the same 2K-token test partition as Mambino reaches 0.120 (Table 1). The protocol is given in footnote § of the same table.

5.2 DEEPMIND MATHEMATICS ARITHMETIC

The reference class is Saxton et al.'s 30M-parameter Transformer at $\sim$76% aggregate on the 56 DM Math modules (Saxton et al., 2019). We evaluate a 515,623-parameter **Mambino** (§4.3, both algorithms) and a 323,609-parameter **Mambino-Residual** μ-refiner (Alg. 1 only). The parser absorbs the linguistic wrapping, so the reducer need only approximate the operator.

Evaluated population. The integer-operand invariant of §4.3 is enforced at sample-collection time by an operand filter, giving acceptance rates ($n = 5000$/module, seed 0) of 2.8% on `arithmetic.mul`, 4.8% on `arithmetic.add_or_sub`, and 97.6% on `arithmetic.add_sub_multiple`. Per-depth acceptance on the last is flat within 1.5 pp across 3–6 operators. The rejection-reason split (non-integer constants vs. magnitude clip) is in Appendix C. Only `add_sub_multiple` therefore has a Saxton et al. number on a comparable population; the `mul` and `add_or_sub` rows in Table 2 keep Saxton's Figure 4 values for reference but are not population-matched.

Training-set overlap. Mambino's DM Math training set is a directly-sampled integer distribution (`build_atomic_reducer_dataset_v10.py`, seed 0) covering 3,043,216 distinct (op, a, b) tuples within $|a|, |b| \leq 999$, which coincides substantially with the filtered population: literal overlap between test tuples and the training key set is 68.8% (`mul`), 51.3% (`add_or_sub`), and 84.0% of chains have every leaf call seen (`add_sub_multiple`). The filtered `mul` space is only 32,209 tuples, so coverage is near-complete by construction. Per-cell accuracies with n on the seen/unseen split of Table 2's Mambino column are in Table 7 (Appendix K); the Sonnet 4.5 column has no training relationship to these samples and is the like-for-like comparison on the unseen partition.

Table 2: Mambino reducer (515,623 params, one checkpoint) under Algorithm 1 hard-argmax recursion on the three DM Math arithmetic modules whose answer format is representation-compatible with Mambino's decimal-position categorical readout (3-way sign softmax plus 10-way per-digit softmax over 12 integer and 6 decimal positions, Section 4.3). Sampling stream: `mathematics_dataset.modules.<module>.train(ef)` with entropy configuration $ef(p) = (3, 10)$; $n = 1000$ per module, seed 0; EM_0 scores a sample correct iff `abs(pred - float(true_value))` rounds to zero, i.e. `int(abs_err == 0)` in the driver (see §5.2 for the evaluated population). [‡] Saxton et al.'s 30M-parameter Transformer values are read from Saxton et al. (2019), Figure 4; that report does not enumerate Figure 4's per-module split or the operand distribution used. [†] The only row directly comparable to ours: `add_sub_multiple` samples integer operands by construction in both settings. Sonnet 4.5 protocol: see footnote [§] of Table 1.

Module	Mambino	Saxton (30M)[‡]	Sonnet 4.5	Notes
`add_or_sub`	0.973	$\sim$0.99	1.000	single-op; not pop.-matched
`add_sub_multiple`[†]	**0.989**	$\sim$0.94	0.445	population-matched
`mul` (depth 1)	**0.917**	$\sim$0.73	1.000	single-product; not pop.-matched

Mambino under Alg. 1. Table 2 lists per-module EM_0 for the Mambino reducer under Algorithm 1 on the `modules.train`(ef) stream with entropy configuration $ef(p) = (3, 10)$, all from the same checkpoint at $n = 1{,}000$ per module.

All three headline modules score at or above $EM_0 = 0.917$.

Mambino under Alg. 2 on ADD chains. On $n = 1000$ `arithmetic__add_sub_multiple` chains (depths $d_2 = 75, d_3 = 527, d_4 = 317, d_5 = 73, d_6 = 8$), the same Mambino reducer under Alg. 2 reaches EM $= 0.986$, close to the 0.989 Alg. 1 number. This confirms that the ADD closed-form rule reproduces integer arithmetic exactly on this operator family.

Mambino-Residual reducer. The 323,609-parameter Mambino-Residual μ-refiner is evaluated on multiplication. On single-product `arithmetic.mul` ($n = 1301$; 112 zero-operand questions short-circuited by the composer, the remaining 1189 all correct) it reaches EM $= \mathbf{1.000}$. On synthetic MUL-only chains of depth 2–5 under a 9-digit product cap ($n = 2000$; per-depth EM $0.734/0.966/0.950/0.988$) it reaches aggregate EM $= \mathbf{0.9095}$. The Mambino reducer under Alg. 2 on the same chains reaches only EM $= 0.622$.

Neither number is attributable to the learned residual. Fixing $\delta \equiv 0$ matches the first and exceeds the second: EM $= 1.000$ on the same 1189 single-product questions and EM $= 1.000$ at every depth $d \in \{2, 3, 4, 5\}$ $(2000/2000)$. Both are consequences of the composer's place-value rule (§4.3): under Alg. 1 every child returns a literal integer, Mambino-Residual sees canonical μ, and the naive digit convolution decodes exactly under the operand cap (Appendix D, `proof-mul-carryfree`).

The residual is designed for the setting the pipeline does not exercise - blurred μ from upstream uncertainty - and the observed 0.734 at depth 2 under the full model rules out the alternative that δ was pushed toward zero by training.

6 Ablations: the reducer is a task-conditional module

Since Algorithms 1 and 2 consume only the reducer's output PMFs, swapping the reducer changes composer accuracy only through its per-node PMFs on the composer-time distribution. For each benchmark we train two reducers under an identical curriculum, differing only in whether the ε-injection branch (§3.2) is active (*Full*) or removed (*Predoff*). On LRA-ListOps the full reducer is $110{,}192$ parameters and Predoff is $101{,}072$ (branch ≈ 9K); on DM Math the full Mambino is $515{,}623$ and Predoff is $450{,}599$ (≈ 65K). Composer code, parser, and evaluation split are byte-identical across each pair; only the loaded checkpoint changes.

6.1 LRA-ListOps: removing the branch improves composition

Figure 4(a,b) visualizes the effect at the reducer level (per-operator + $294{,}843$-call weighted aggregate) and at the composer level under both algorithms. Predoff wins under both algorithms ($+4.45$ pp Alg. 1, $+1.60$ pp Alg. 2), despite marginally lower accuracy on the atomic-reducer curriculum's held-out split (0.9986 vs 0.9992). The full breakdown is in Table 5 (Appendix K).

Where the gap comes from. Per-node logging localizes the gap to `SM` mod 10 ($+7.17$ pp) and to short subtrees, concentrated at low N and shrinking as N grows. The per-operator, per-N, per-depth, and error-mode breakdowns are in Table 4 (Appendix J).

Training overlap on LRA is a null. Measured against the LRA-ListOps Mambino's training partition (alias `eval-A35z`), Full-unseen accuracy is 0.9284 and Full-seen is 0.9167; unseen accuracy is marginally *higher* than seen, and $1{,}999$ of the $2{,}000$ trees carry at least one unseen leaf call. The 0.9535 headline is already a held-out number. The single LRA cell in which Full beats Predoff is $N=2$ atomic on unseen arguments (Full 0.8215 vs. Predoff 0.7575, $+6.4$ pp, Newcombe 95% CI $[+0.0413, +0.0865]$). That cell is the LRA subset structurally closest to DM Math: an operator applied to arguments the reducer has not stored, at the shortest subtree length, where the reducer is furthest from ceiling (Full-seen at $N=2$ is 0.6953, the worst cell in the LRA table). The rule is the same across both benchmarks: the branch contributes where the operator stresses the reducer and the input is novel, and nothing where the reducer is at ceiling.

6.2 DM Math: the branch's contribution is on the unseen partition

The same ablation on the DM Math Mambino reducer reverses direction: Predoff loses on every arithmetic module we test (Figure 4(c,d), Table 6 in Appendix K).

On the seen/unseen partition of an independent deterministic draw of $n=1000$/module from the same stream (pooled accuracies agree with Tables 2 and 6 within 0.8 pp; Table 7 in Appendix K, twelve cells with $n_{\mathrm{s}}/n_{\mathrm{u}}$; Wilson 95% CIs in `eval-A31z`–`eval-A33z`), seen-partition Full$-$Predoff gaps are 2.8 pp or less: within CI resolution on two of three modules, and significant on `mul` but an order of magnitude below its unseen counterpart. On unseen inputs the branch is worth $+\mathbf{25.0}$ pp on `mul`, $+\mathbf{11.3}$ pp on `add_sub_multiple`, and $+\mathbf{5.1}$ pp on `add_or_sub`, ordered by the amount of positional carry each operator requires. The predictor does not improve the reducer's capacity to *store* operator behavior; it improves *computation* on inputs the reducer has not stored, and the pooled Alg. 2 gap of $+16.3$ pp on `add_sub_multiple` chains therefore *widens* once contamination is controlled for.

Parameter-count and algorithm-invariance. Predoff carries $\sim$8–13$\%$ fewer parameters than Full (Table 8), but the direction reversal across benchmarks rules out a capacity-only reading. Under Alg. 2 the `add_sub_multiple`-chain gap widens further (Full $0.989 \to 0.986$, Predoff $0.973 \to 0.823$). The extra Predoff loss reveals a second effect of the branch: it sharpens the softmax around the correct answer as well as improving the argmax.

6.3 PMF-layer decoupling and the operator dichotomy

The decoupling holds across reducer architectures, not only by construction. An N-ary Tree-LSTM cell (Tai et al., 2015) swapped for the SSM reducer at 112K vs. 110K parameters and retrained under

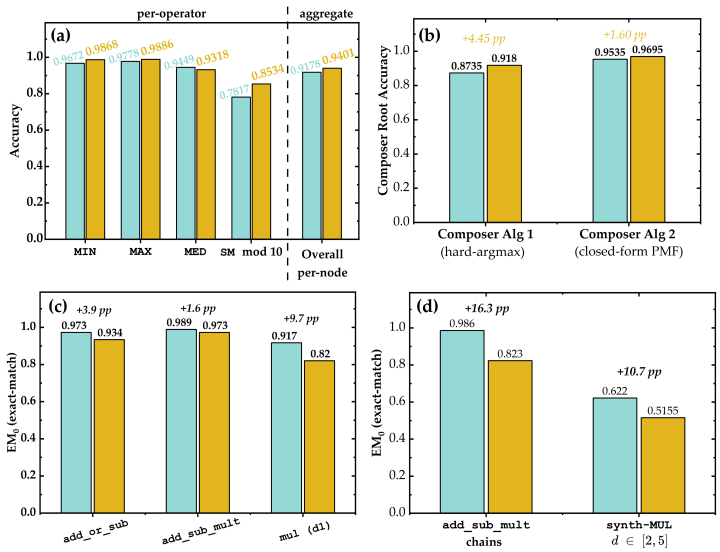


Figure 4: Predictor-branch ablation, Full (teal) vs Predoff (gold). **(a)** LRA-ListOps reducer accuracy per operator + aggregate. **(b)** LRA-ListOps composer root accuracy. **(c)** DM Math Alg. 1 per-module EM. **(d)** DM Math Alg. 2 on `add_sub_multiple` and synth-MUL chains. LRA-ListOps (a, b) rewards removing the branch; DM Math (c, d) rewards keeping it.

the Mambino curriculum reaches 0.7445 under Alg. 1 and 0.9260 under Alg. 2 (Table 1, † rows), within 2.75 pp of the SSM reducer under Alg. 2.

The seen/unseen split maps onto two operator regimes, *saturating* for LRA-ListOps and *carry-propagating* for DM Math arithmetic. The branch is a compensating mechanism only in the second, consistent with the diagonal-scan TC^0 picture the reducer's architecture inherits (§2) (Merrill et al., 2024), which bounds per-token state complexity.

7 Conclusion

Parser-supplied structure with a closed-form PMF composer and a small state-space reducer reaches $\mathbf{0.9535}$ on LRA-ListOps and $\mathbf{0.989}$ on `arithmetic.add_sub_multiple` at $58\times$ fewer parameters than the Saxton reference class. Partitioning each evaluation by training overlap characterizes the reducer's predictor branch cleanly: it contributes 2.8 pp or less on operator calls the reducer has seen, and up to 25.0 pp on unseen carry-propagating calls, with the effect graded by how much positional carry an operator demands. Reducer and composer meet only through PMFs, so either can be swapped or retrained without touching the other; a Tree-LSTM reducer at matched parameter budget slots into the same composer and lands within 2.75 pp of the SSM reducer under Alg. 2. This *PMF-layer decoupling* is what makes the framework applicable beyond the two benchmarks reported here (Appendices E–F).

Reproducibility Statement

Every numerical claim in this paper is anchored to a specific released checkpoint and a specific released evaluation artifact. The full paper-claim $\rightarrow$ checkpoint alias $\rightarrow$ evaluation-artifact alias mapping is given in Appendix L. The complete inference-only reproduction API and all checkpoints will be deposited to an institutional DOI-issuing repository at publication. The anonymized mirror at `anonymous.4open.science/r/mambino-iclr-release-9018` ships the cross-architecture N-ary Tree-LSTM reducer and its Alg. 1 / Alg. 2 composed-evaluation script (aliases `eval-A22`, `eval-A23`). The full artifact bundle (`eval-A01`–`eval-A37z`, all six checkpoints with SHA-256 digests, the runner scripts for the post-hoc analyses, and the machine-checkable proof `proof-mul-carryfree`) is staged and available to reviewers through the OpenReview supplementary channel.

USE OF AI ASSISTANCE

We used LLM-based coding and writing assistants (Anthropic Claude family models) throughout this project. For the *implementation*, assistants helped author and refactor model code (the real-diagonal state-space block, the predictor branch, the closed-form PMF composers), debug training runs, write evaluation harnesses, and cross-check the reference implementations of the closed-form PMF rules against their mathematical specifications. For the *write-up*, assistants drafted and rewrote passages under the authors' direction in the Introduction, Related Work, Results, Ablations, and Appendices; cross-checked numerical claims against machine-generated evaluation artifacts; verified that the pseudocode in §4 matches the released code; checked reference metadata and searched for naming collisions; and performed spelling and consistency passes. Assistants also helped formalize the carry-free decode identity of Appendix D as a Lean 4 proof and derive the coefficient bound stated there; the identity and the proof strategy were the authors'. Research ideation - the parser-plus-reducer-plus-closed-form-composer framework, the PMF-layer decoupling property, the predictor-branch architecture, the saturating-vs-carry-propagating operator dichotomy, and every hypothesis tested - was the authors' alone. No LLM was used to generate training or evaluation data.

REFERENCES

Kareem Ahmed, Stefano Teso, Kai-Wei Chang, Guy Van den Broeck, and Antonio Vergari. Semantic probabilistic layers for neuro-symbolic learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. A fast unified model for parsing and sentence understanding. In *Proc. Association for Computational Linguistics (ACL)*, 2016. URL https://arxiv.org/abs/1603.06021.

Jishnu Ray Chowdhury and Cornelia Caragea. Modeling hierarchical structures with continuous recursive neural networks. In *Proc. International Conference on Machine Learning (ICML)*, 2021. URL https://arxiv.org/abs/2106.06038.

Jishnu Ray Chowdhury and Cornelia Caragea. Beam tree recursive cells. In *International Conference on Machine Learning (ICML)*, 2023.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024. URL https://arxiv.org/abs/2405.21060.

Lennert De Smet and Pedro Zuidberg Dos Martires. A fast convoluted story: Scaling probabilistic inference for integer arithmetic. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Harriet Feldman and Karl J. Friston. Attention, uncertainty, and free-energy. *Frontiers in Human Neuroscience*, 4:215, 2010.

Yang Feng. fengyang0317/listops-1000: A LRA-style ListOps subset on hugging face. Hugging Face dataset, 2024. URL https://huggingface.co/datasets/fengyang0317/listops-1000.

Karl Friston. The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, 11: 127–138, 2010. doi: 10.1038/nrn2787. URL https://doi.org/10.1038/nrn2787.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023. URL https://arxiv.org/abs/2312.00752.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. HiPPO: Recurrent memory with optimal polynomial projections. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://arxiv.org/abs/2008.07669.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *Proc. International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2111.00396.

Yoon Kim, Alexander Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. Unsupervised recurrent neural network grammars. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, 2019.

Ziyang Li, Jiani Huang, and Mayur Naik. Scallop: A language for neurosymbolic programming. In *Programming Language Design and Implementation (PLDI)*, 2023.

William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. In *Proc. International Conference on Learning Representations (ICLR)*, 2017. URL https://arxiv.org/abs/1605.08104.

Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. Mega: Moving average equipped gated attention. In *Proc. International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2209.10655.

Andreas Madsen and Alexander Rosenberg Johansen. Neural arithmetic units. In *Proc. International Conference on Learning Representations (ICLR)*, 2020. URL https://arxiv.org/abs/2001.05016.

Jean Maillard, Stephen Clark, and Dani Yogatama. Jointly learning sentence embeddings and syntax with unsupervised Tree-LSTMs. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL)*, 2017.

Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. DeepProbLog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *International Conference on Learning Representations (ICLR)*, 2019.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In *International Conference on Machine Learning (ICML)*, 2024.

Beren Millidge, Anil Seth, and Christopher L. Buckley. Predictive coding: a theoretical and experimental review. *arXiv preprint arXiv:2107.12979*, 2022. URL https://arxiv.org/abs/2107.12979.

Pablo Miralles-González, Javier Huertas-Tato, Alejandro Martín, and David Camacho. On the locality bias and results in the long range arena. *arXiv preprint arXiv:2501.14850*, 2025. URL https://arxiv.org/abs/2501.14850.

Maxwell Nye, Armando Solar-Lezama, Joshua B. Tenenbaum, and Brenden M. Lake. Learning compositional rules via neural program synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Rajesh P. N. Rao and Dana H. Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1):79–87, 1999. doi: 10.1038/4580. URL https://doi.org/10.1038/4580.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *Proc. International Conference on Learning Representations (ICLR)*, 2019. URL https://arxiv.org/abs/1904.01557.

Yikang Shen, Shawn Tan, Alessandro Sordoni, Peng Li, Jie Zhou, and Aaron Courville. Ordered memory. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Jimmy T. H. Smith, Andrew Warrington, and Scott W. Linderman. Simplified state space layers for sequence modeling. In *Proc. International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2208.04933.

Taylan Soydan, Nikola Zubić, Nico Messikommer, Siddhartha Mishra, and Davide Scaramuzza. S7: Selective and simplified state space layers for sequence modeling. *arXiv preprint arXiv:2410.03464*, 2024. URL https://arxiv.org/abs/2410.03464.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proc. Association for Computational Linguistics (ACL)*, 2015. URL https://arxiv.org/abs/1503.00075.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. In *Proc. International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2011.04006.

Andrew Trask, Felix Hill, Scott E. Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. In *Proc. Neural Information Processing Systems (NeurIPS)*, 2018. URL https://arxiv.org/abs/1808.00508.

Daniel M. Wolpert and Zoubin Ghahramani. Computational principles of movement neuroscience. *Nature Neuroscience*, 3(Suppl 11):1212–1217, 2000.

A Base state-space block: self-contained reference

For readers less familiar with the S5 lineage (Smith et al., 2023), we give a self-contained reference for the three components §3.1 attributes to established practice.

Base recurrence. Each layer is a discrete-time real-diagonal state-space model with hidden state $h_t \in \mathbb{R}^P$:

$$h_t = A\, h_{t-1} + B\, x_t, \tag{5}$$

$$y_t = C\, h_t, \tag{6}$$

where $A = \mathrm{diag}(a_1, \ldots, a_P)$ with $a_p \in (\lambda_{\min}, \lambda_{\max}) \subset (0, 1)$, $B \in \mathbb{R}^{P \times D_{\text{in}}}$, and $C \in \mathbb{R}^{D_{\text{out}} \times P}$. We follow the real-diagonal simplification of S5 (Smith et al., 2023) of the earlier complex-conjugate S4 formulation (Gu et al., 2022).

Eigenvalue parameterization. Each diagonal entry is written as $a_p = \lambda_{\min} + (\lambda_{\max} - \lambda_{\min})\, \sigma(\tilde{a}_p)$ with $\tilde{a}_p \in \mathbb{R}$ the trained parameter and σ the logistic sigmoid, which keeps a_p inside the stable range without projection. Throughout the paper the bounds are initialized at $\lambda_{\min} = 10^{-3}$ and $\lambda_{\max} = 0.9995$ and learned per layer (Appendix G), corresponding at initialization to per-token decay timescales $\tau_p = -1/\log a_p$ spanning roughly one to ~ 2000 tokens. Initialization places the $\tilde{a}_p$ so the resulting τ_p are log-uniformly spaced across the bank; A is initialized for coverage of the full timescale range, not from any HiPPO prescription (Gu et al., 2020).

M-gated error injection. The predictor branch of §3.2 produces a per-token error $\varepsilon_t \in \mathbb{R}^{D_{\text{in}}}$ and a mask $M_t = \sigma\big(g_\varepsilon(\tilde{x}_t)\big) \in [0, 1]^{D_{\text{in}}}$, and the gated error $W_\varepsilon\,(M_t \odot \varepsilon_t)$ is added to the projected input before the base recurrence integrates it. At $M_t = 0$ the branch is effectively removed; at $M_t = 1$ the full error is injected. This pathway is what the ablations of §6.1 (-4.45 pp) and §6.2 ($+16.3$ pp) switch on and off.

Training objective. Every reducer is trained on isolated operator calls (no trees seen at training time). The target is the operator's output token, represented as either the categorical readout of Mambino (both LRA-ListOps and DM Math variants) or the residual μ-target of Mambino-Residual (§4.3). The loss is standard supervised cross-entropy for the categorical readouts and mean-squared error for the μ-residual, averaged over the batch. Batch size, learning rate schedule, optimizer, and per-benchmark hyperparameters are in Appendix G.

B Closed-form PMF rules for the four LRA-ListOps operators

Section 4.2 describes the four operator rules in prose; here are the full statements referenced from the main body as Equations (7)–(10). Each rule is stated for k independent child PMFs $X_1, \ldots, X_k$ over the alphabet $\{0, \ldots, 9\}$ with output Y.

- `MIN`: the survival function of the minimum of independent random variables is the product of their survival functions,

$$P(Y \geq y) = \prod_{i=1}^{k} P(X_i \geq y). \tag{7}$$

- `MAX`: the cumulative distribution function of the maximum is the product of the individual CDFs,

$$P(Y \leq y) = \prod_{i=1}^{k} P(X_i \leq y). \tag{8}$$

- `SM` mod 10: the sum-modulo-10 operator propagates as a circular convolution over the residue ring $\mathbb{Z}/10\mathbb{Z}$,

$$p_Y[y] = \big(p_{X_1} \circledast \cdots \circledast p_{X_k}\big)[y \bmod 10]. \tag{9}$$

- `MED`: the median of a fixed-size list is computed by joint enumeration over child support, weighted by the joint probability of each outcome tuple,

$$p_Y[y] = \sum_{\mathbf{x}\,:\,\mathrm{median}(\mathbf{x})=y} \prod_{i=1}^{k} p_{X_i}[x_i]. \tag{10}$$

  For even k we follow the LRA convention $\mathrm{median}(\mathbf{x}) = \big\lfloor \big(x_{(k/2)} + x_{(k/2+1)}\big)/2 \big\rfloor$, the integer floor of the mean of the two central order statistics $x_{(i)}$, so the median is single-valued and p_Y sums to one. Naive enumeration is $O(10^k)$; the released composer (`pmf_med`) enumerates only over the *uncertain* arguments, collapsing any child with $\max p > 0.9999$ to its argmax value, and falls back to a 10,000-sample Monte-Carlo estimator when the uncertain-dim exceeds 10^6 combinations. For typical LRA-ListOps nesting (one or two uncertain arguments per node) this stays at 10–100 combinations.

The rules are implemented in the LRA-ListOps composer shipped in the anonymized code mirror. On one-hot inputs each rule reduces to the exact integer operator; the DM Math rules are additionally verified by the one-hot self-test described alongside them in Appendix D.

C DM MATH DIGIT REPRESENTATION AND REDUCER VARIANTS

Full definitions for §4.3. DM Math operands and answers are decimal digit sequences with an explicit sign, $x = s \cdot \big(\sum_{k=0}^{K_{\mathrm{int}}-1} d_{\mathrm{int}}^{(k)} \cdot 10^k + \sum_{k=1}^{K_{\mathrm{dec}}} d_{\mathrm{dec}}^{(k)} \cdot 10^{-k}\big)$ with $s \in \{-, 0, +\}$ and each $d \in \{0, \ldots, 9\}$; throughout the paper we use $K_{\mathrm{int}} = 12$ integer digits and $K_{\mathrm{dec}} = 6$ decimal digits ($K = 18$ digit positions plus one sign position). The two DM Math reducers share only the SSM trunk of Section 3; they differ substantively in input, readout, and training objective.

Rejection-reason split of the operand filter (referenced from §5.2). On `arithmetic.mul` and `arithmetic.add_or_sub`, roughly 85% of the samples rejected by the filter are non-integer constants emitted by the DM Math module `sample_integer_or_decimal` with its fixed integer-vs-decimal mixing probability; the residual $\sim 15\%$ are integer samples whose operand or answer exceeds the $|\cdot| \leq 999$ magnitude clip. On `arithmetic.add_sub_multiple`, the module emits integer operands by construction, so all 2.4% of rejections come from the magnitude clip; this is also why per-depth acceptance stays flat within 1.5 pp across chains of 3–6 operators.

Mambino takes the token stream directly and places a categorical softmax at every output position: a 3-way softmax at the sign position and a 10-way softmax at each of the $K = 18$ digit positions. Because every position emits a genuine probability distribution, Mambino is composable under both Algorithm 1 (argmax at each position gives the integer answer) and Algorithm 2 (per-position PMFs propagate through the closed-form operator rules below).

Mambino-Residual is architecturally different, and uses its own internal per-decade lattice with $K^{\mathrm{D}} = 13$ decades ($K_{\mathrm{int}}^{\mathrm{D}} = 9$ integer, $K_{\mathrm{dec}}^{\mathrm{D}} = 4$ decimal) and shift $\Delta^{\mathrm{D}} = 4$; the constants $(K^{\mathrm{D}}, K_{\mathrm{int}}^{\mathrm{D}}, K_{\mathrm{dec}}^{\mathrm{D}}, \Delta^{\mathrm{D}})$ are Mambino-Residual-specific and differ from Mambino's $(18, 12, 6, 6)$. It is a *residual μ-refiner*: instead of consuming the two operands directly, its input is the per-decade *symbolic naive product* of the two operand μ-vectors,

$$\mathrm{naive_mul}^{(k)} = \sum_{\substack{i+j-\Delta^{\mathrm{D}}=k \\ 0 \leq i,j < K^{\mathrm{D}}}} \mathrm{op1_mu}^{(i)} \cdot \mathrm{op2_mu}^{(j)}, \tag{11}$$

a fixed pointwise convolution over per-decade means (no learned parameters; the position offset $\Delta^{\mathrm{D}} = K^{\mathrm{D}}_{\mathrm{dec}} = 4$ re-anchors the product so that decade index $k = 0$ denotes the 10^{-4} place, matching the Mambino-Residual operand indexing). The SSM trunk emits a per-decade correction $\delta^{(k)}$ and the final readout is

$$\widehat{\mu}^{(k)} \;=\; \mathrm{naive_mul}^{(k)} + \delta^{(k)}. \tag{12}$$

The head is zero-initialized so that $\delta^{(k)} \equiv 0$ and $\widehat{\mu}^{(k)} = \mathrm{naive_mul}^{(k)}$ at step 1, and the network only learns the denoising correction. Training minimizes $L_{\mathrm{task}} = \|\widehat{\mu} - \mu_{\mathrm{target}}\|_1$ (per-position L1) plus an intrinsic $\lambda\,\mathbb{E}[\varepsilon^2]$ term at $\lambda = 1.0$; the final answer decodes as $Y = \sum_k 10^{k - K^{\mathrm{D}}_{\mathrm{dec}}} \cdot \widehat{\mu}^{(k)}$ with no learned converter and no digit-bucket rounding. Because the readout is a per-decade mean rather than a distribution, Mambino-Residual is composable under Algorithm 1 only. The design rationale is that fitting a small residual over an algebraically correct baseline is a lower-capacity target than fitting the whole product with a categorical readout; under the canonical-μ inputs that Alg. 1 supplies at inference the baseline is already exact (§5.2, and `proof-mul-carryfree` below), so the pipeline evaluated here does not exercise the residual's design regime.

Closed-form propagation for Mambino under Algorithm 2. Each digit position propagates by a linear convolution over child PMFs whose mod-10 marginal gives the digit and floor-division marginal gives the carry into the next position; signed operands are handled by a 3×3 sign-pair enumeration; MUL uses a per-place-value partial-product enumeration over the 10×10 digit-pair support followed by carry resolution on the shifted grid. Full statements of all three rules are in Equations (13)–(17) of Appendix D. Position-wise independence is assumed at each step, and sign–magnitude joint correlations are marginalized after each operator application, introducing a controlled approximation whose empirical cost is quantified in Section 5.2.

D Closed-form PMF rules for signed addition and multiplication

Section 4.2 of the main body states the shape of the closed-form propagation for Mambino under Algorithm 2. Full equations follow.

Unsigned addition: linear convolution, digit marginal, and carry. For an addition chain $Y = X_1 + X_2 + \cdots + X_n$ of Mambino operands, each digit position k first forms the PMF of the *linear* (non-wrapped) sum of the n same-position operand digits, supported on $\{0, \ldots, 9n\}$, then folds in the incoming carry PMF from position $k-1$,

$$q^{(k)} = p^{(k)}_{X_1} * p^{(k)}_{X_2} * \cdots * p^{(k)}_{X_n}, \qquad r^{(k)} \;=\; q^{(k)} * p^{(k)}_{\mathrm{carry}}, \tag{13}$$

$$p^{(k)}_{Y}[y] = \sum_{m\,:\,m \bmod 10 = y} r^{(k)}[m], \qquad p^{(k+1)}_{\mathrm{carry}}[c] = \sum_{m\,:\,\lfloor m/10 \rfloor = c} r^{(k)}[m], \tag{14}$$

where $*$ denotes linear (acyclic) convolution and $p^{(0)}_{\mathrm{carry}} = \delta_0$: the digit PMF at position k is the mod-10 marginal of the carry-included sum $r^{(k)}$, and the outgoing carry PMF is its floor-division-by-10 marginal.

Signed addition: 3×3 sign-pair enumeration. The signed sum $Y = X_1 + X_2$ with X_i carrying a sign–magnitude joint PMF $p_{X_i}(s_i, |x_i|)$ marginalizes as

$$\begin{aligned} p_Y(s, |y|) \;=\; & \sum_{s_1, s_2 \in \{-,0,+\}} \; \sum_{|x_1|, |x_2|} p_{X_1}(s_1, |x_1|)\, p_{X_2}(s_2, |x_2|) \\ & \cdot \mathbb{1}[\mathrm{SignRule}(s_1, s_2) = s \;\wedge\; \mathrm{MagRule}(s_1, s_2, |x_1|, |x_2|) = |y|]\,, \end{aligned} \tag{15}$$

where $\mathrm{SignRule}$ and $\mathrm{MagRule}$ implement the usual case-split over the nine (3×3) sign-pair combinations: same nonzero signs $\Rightarrow$ magnitudes add; opposite signs $\Rightarrow$ smaller magnitude subtracted from larger with Y taking the sign of the larger magnitude (equal magnitudes canceling to sign 0); a sign-0 operand passes the other through unchanged. This mixed-sign branch lets Equations (13)–(14) cover `arithmetic__add_sub_multiple` with no separate subtraction rule.

Multiplication: per-place-value partial products. For MUL the closed-form rule is a per-place-value partial-product enumeration over the 10×10 digit-pair support followed by a carry-resolution

over the shifted grid,

$$p^{(i,j)}_{\text{partial}}[z] = \sum_{\substack{a,b\in\{0,\ldots,9\}\\ a\cdot b=z}} p^{(i)}_X[a]\cdot p^{(j)}_Y[b], \quad z\in\{0,\ldots,81\}, \tag{16}$$

$$p^{(k)}_{Y_{\text{prod}}} = \text{ResolveCarry}\left(\mathop{\ast}_{i+j=k} p^{(i,j)}_{\text{partial}}\right), \tag{17}$$

where $*$ denotes the position-wise convolution combining the per-position partial-product PMFs along the anti-diagonal $i+j=k$, under position-wise reducer-output independence (each partial product treated as an independent factor). Under one-hot operands the rule recovers exact integer multiplication; under calibrated softmax inputs it carries the reducer's per-position uncertainty upward at a controlled approximation quantified in Section 5.2.

One-hot self-test. The bit-exactness claim for the DM Math ADD/SUB/MUL family is verified by a deterministic self-test: 5000 uniformly-sampled operator calls covering the five operators `ADD`, `MUL`, signed `ADD`, signed `SUB`, and signed `MUL` (1000 one-hot operand tuples per operator); each composed output is compared bit-exactly against reference integer arithmetic. Result: $5000/5000$ exact (alias `eval-A36z` in Table 9; the rule implementation and the self-test runner ship in the artifact bundle and run standalone).

Carry-free decode of `naive_mul` (alias `proof-mul-carryfree`). Equation (11)'s naive_mul is the polynomial product of the two digit polynomials evaluated at base 10: for integer operands a, b with digit vectors x_i, y_j, $\sum_k \text{naive_mul}^{(k)}\cdot 10^{k-\Delta^{\text{D}}} = \left(\sum_i x_i 10^i\right)\left(\sum_j y_j 10^j\right) = ab$, identically in the coefficient values. Carry resolution re-quantizes coefficients into $\{0,\ldots,9\}$ and is never required by decoding, since the weighted sum is invariant under whether a coefficient of value 81 sits at decade s or splits into $(1)\cdot 10^s + (8)\cdot 10^{s+1}$. Under Mambino-Residual's 9-digit product cap ($ab < 10^9$), $\text{naive_mul}^{(k)} = 0$ for k outside the K^{D}-window; for operands with m and n digits ($m+n \le 10$ under the cap) the largest possible per-decade coefficient is $9^2\cdot\min(m,n) \le 405$; all partial sums stay far below 2^{53}, so float64 evaluation of the decode is exact. We release a Lean 4 kernel proof of the identity as `proof-mul-carryfree`: theorem `eval_conv : eval (conv xs ys) = eval xs * eval ys` under `eval (d :: ds) = d + 10 * eval ds` over `Nat`, proved by structural induction on the digit list (lemmas `eval_addP`, `eval_smul`), with two kernel-reduced `by decide` spot checks; no `mathlib` import. The identity holds for arbitrary natural-number coefficients, so it covers any operand distribution the reducer can present under Alg. 1's integer-invariant regime; we do *not* claim bit-exactness of multiplication in general, nor under non-canonical (decimal or blurred) μ.

E Design-level predictions of the framework

The characterization of §6.3 makes two design-level predictions about untested tasks. (i) The reducer's parameter budget scales with residual operator complexity (the part of the primitive that does not close in a hand-writable rule), not with task expression length. (ii) The predictor branch is load-bearing iff the primitive's per-token intermediate state is positional with long-range carry. Both are testable at the scale of a single reducer training run: set-operation composition should saturate at Mambino size with no branch, whereas floating-point arithmetic over k significant digits should show a MUL-analogous positional-depth gap without it. Since Algorithms 1 and 2 consume only the reducer's output PMF, the reducer class is also interchangeable: a Tree-LSTM cell, a small Transformer block, or a Mamba block with a categorical head at its output layer would all fit. The framework's contribution is not the SSM choice (Smith et al., 2023) but the isolation of PMF-layer decoupling as the property that makes reducers interchangeable.

Retrospective on Mambino-Residual for capped MUL. Prediction (i) says the reducer's residual capacity should scale with the part of the primitive not expressed by the composer's closed-form rule. Multiplication under Mambino-Residual's 9-digit product cap has none: on canonical μ the naive digit convolution decodes to ab exactly (Appendix D, alias `proof-mul-carryfree`), so the prediction's zero-complexity limit is $\delta \equiv 0$. The $\delta \equiv 0$ run (§5.2, aliases `eval-A08z`, `eval-A09z`) verifies that limit and, in the same motion, tells us the current pipeline does not exercise the residual's design regime: canonical μ is a special case, and δ's value proposition survives only once the reducer receives blurred μ or a per-decade PMF. The criterion in prediction (i) was stated correctly; we simply did not apply it to capped MUL, and the $\delta \equiv 0$ evaluation is what caught the omission.

F Applicability of the parse-then-compose architecture beyond compositional reasoning

The composition pattern described in Section 4 is neither task-specific nor benchmark-specific. Its structural requirements are: (i) a compositional scaffold that is either analytically available or recoverable from a structural prior, and (ii) a primitive operation at each node that is well-defined per-call but is either analytically hard to express under noisy operands or generalizes poorly across a wide input distribution. Below we sketch four problem classes where the pattern applies, all of which are outside the direct scope of this paper.

F.1 Time-varying prediction as a Kalman-filter analogue

The internal-state predictor branch of Section 3.2 is structurally a discrete-time Kalman-filter update: the main recurrence carries the state estimate, the predictor branch emits the innovation term $\varepsilon_t = x_t - \hat{x}_t$, and W_ε acts as a learned Kalman gain. The identification is not just formal. The same reducer class, trained under the same objective, generalizes to time-varying prediction tasks where the underlying dynamics are partially known: the known state-space model of the dynamics plays the parser role, and a residual learner over the unmodeled component plays the reducer role.

F.2 Dynamical-system identification with structural priors

System identification in engineering typically factorizes into a structural prior (from first-principles physics) and a residual learner that captures unmodeled effects. The parse-then-compose architecture is the same factorization, with the physics-based ODE integrator playing the role of the composer.

F.3 Motor control and cerebellar forward models

The predictor branch's structure parallels the forward-model / efference-copy loops in the cerebellum (Wolpert & Ghahramani, 2000), where an expected sensory consequence of a motor command is subtracted from the actual sensory input to produce a corrective residual. A reducer with such a branch is a candidate building block for motor-control policies that must compose learned primitives (reach, grasp) under a task-level symbolic plan.

F.4 Edge inference with compute-in-memory substrates

Because the SSM reducer's per-timestep compute is bounded and its parameters are static, the reducer is amenable to deployment on compute-in-memory substrates. The parser and recursive evaluation loop remain on a digital host; only the reducer's SSM scan runs on the analog substrate. The partition matches the per-component energy footprint: the parser is fast and control-flow-heavy (well-suited to a digital host), while the reducer is compute-heavy per invocation and repeated many times per expression (well-suited to a low-energy analog substrate). Full characterization of the reducer on a specific substrate is out of scope for this paper.

G Training recipe and dataset construction

G.1 Base-block A parameterization and per-reducer constants

The diagonal $A \in \mathbb{R}^{d_C}$ of the base-block recurrence (Equation 1) is parameterized as

$$A = \lambda_{\min} + (\lambda_{\max} - \lambda_{\min}) \cdot \sigma(A_{\text{raw}}), \tag{18}$$

with $\lambda_{\min} = \sigma(\lambda_{\text{lo,raw}})$ and $\lambda_{\max} = \lambda_{\min} + \text{softplus}(\lambda_{\text{width,raw}}) \cdot (1 - \lambda_{\min})$ all learned per layer. The sigmoid bounds $\lambda_{\min}$ but the softplus does not hard-clamp $\lambda_{\max}$ at 1; in the released checkpoints $\lambda_{\max}$ drifted to at most 1.0004, and the largest trained eigenvalue a_p is 0.970, so every state dimension has a real eigenvalue in $(0, 0.97]$. Concrete state dimensions used in the paper: $d_C = 96$ for the LRA-ListOps Mambino (single-block, $L = 1$), $d_C = 128$ for the DM Math Mambino ($L = 4$), and $d_C = 32$ for Mambino-Residual ($L = 2$).

G.2 Optimization

Both Mambino reducers are trained with AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$), decoupled weight decay 0.05, a single learning-rate schedule per phase, and gradient clipping at 1000; Mambino-Residual uses the same optimizer with gradient clipping at 2500 (§G.5). Batch size, per-phase peak learning rates, and warmup differ between reducers; values below record the training-time configuration used to produce each released checkpoint. The training pipeline itself is not part of the release, which distributes frozen checkpoints, archived evaluation artifacts, and inference-time evaluators only.

G.3 CURRICULUM: LRA-LISTOPS MAMBINO

Architecture: stacked Mambino block with $L=1$ layer, $d=128$, $d_C=96$, vocabulary size 19 (LRA-ListOps digits + operators + brackets + PAD). Training runs four cumulative curriculum phases of 50 epochs each (200 epochs total) on atomic single-reducer expressions extracted from LRA-ListOps: (P1) $N \in \{2,3\}$; (P2) $N \in \{2..5\}$; (P3) $N \in \{2..7\}$; (P4) $N \in \{2..10\}$, where N is the number of operator arguments. AdamW stays warm across phases, with a fresh cosine schedule per phase (5-epoch warmup) and stair-stepped peak LR $1\times10^{-3} \to 5\times10^{-4} \to 3\times10^{-4} \to 2\times10^{-4}$, cosine to global floor 1×10^{-4} within each phase. Batch size 256, $\lambda_{\mathrm{pc}} = 0.1$, predictor branch weight decay 1×10^{-6} (main SSM weight decay 0.05).

G.4 CURRICULUM: DM MATH MAMBINO

The architecture is a stacked Mambino block with $L = 4$ layers, $d = 128$, $d_C = 128$, and a wide-representation input/output of 60 input tokens and 19 output positions (1 sign + 12 integer digits + 6 decimal digits). Training runs four curriculum phases at $30/45/70/105$ epochs on 1-digit through 4-digit operands respectively, with all operator mixups (ADD, SUB, MUL, DIV) present from Phase 1. Batch size 1024, peak LR 1×10^{-3}, warmup fraction 10%, cosine to floor 1×10^{-4} within each phase, weight decay 0.05, gradient clip 1000, $\lambda_{\mathrm{pc}} = 0.1$. The readout is a categorical softmax at each of the 19 output positions.

G.5 CURRICULUM: MAMBINO-RESIDUAL (DM MATH MUL ONLY)

Mambino-Residual is architecturally distinct from Mambino: it is a residual μ-refiner (Section 4.3, Equation (12)). The trunk is a Mambino block ($d=64$, $d_C=32$, $L=2$, decade embedding 8-dim, additive PC, parallel scan, all internal gates {eps, input, output} on, GELU readout) whose UNARY per-step encoder consumes the per-decade symbolic naive product $\mathrm{naive_mul}$ of the two operand μ-vectors (a fixed convolution with no learned parameters). The per-step head emits a per-decade correction δ (zero-initialized, so $\widehat{\mu} = \mathrm{naive_mul}$ at step 1), yielding the final readout of Equation (12). Training loss is $L = L_{\mathrm{task}} + \lambda \cdot \mathrm{mean}(\varepsilon^2)$ with $\lambda = 1.0$ (*note*: Mambino-Residual uses a larger intrinsic-loss weight than the other reducers), where L_{task} is a mask-averaged L1 between $\widehat{\mu}$ and the target μ-vector under a structural-zero mask derived from $\mathrm{target_int} = \mathrm{op1} \cdot \mathrm{op2}$. We optimize with AdamW under the same β and weight-decay defaults as the other reducers, gradient clip 2500, cosine LR schedule to floor 1×10^{-5}, save-best on μ-MAE-significant, and early exit at 1×10^{-4}, step-based to 386k steps. The final answer decodes as $Y = \sum_k 10^{k-K^{\mathrm{D}}_{\mathrm{dec}}} \cdot \widehat{\mu}^{(k)}$ with no learned converter and no digit-bucket rounding.

G.6 DATASET CONSTRUCTION

LRA-ListOps uses the standard $96{,}000/2{,}000/2{,}000$ split (Tay et al., 2021) at $T \leq 2048$ tokens. Atomic single-reducer training data is extracted from LRA training expressions; DM Math training data is generated by the official DM Math templates (Saxton et al., 2019). Synthetic MUL-only depth-2 through depth-5 chains are generated programmatically with a 9-digit product cap; per-depth sample counts appear in the results tables of Section 5.2.

G.7 RELEASE CONTENTS AND ALIAS SCHEME

Every numerical claim in the main body is anchored to a specific released checkpoint and a specific released evaluation artifact. Checkpoints will be deposited to an institutional DOI-issuing repository at publication; the accompanying code release (checkpoint loading, inference under both composition algorithms, and per-module evaluation) allows independent regeneration of every value reported. During review, each value is available as an archived evaluation artifact anchored to the SHA-256 of the checkpoint that produced it; the artifacts that can be regenerated without the reducer package (the Tree-LSTM ablation, the zero-shot LLM baselines, the one-hot self-test, and the Lean proof) ship with their runners. The per-claim alias mapping is Appendix L below. Release aliases cover the paper's numerical claims and include suffix-`z` aliases for artifacts added after the initial evaluation pass (`eval-A01`–`eval-A37z`, plus `proof-mul-carryfree`). Alias numbering is not contiguous; `eval-A06` and `eval-A19` are not part of this release.

Implementation notes on Algorithm 1. Three implementation choices in the released Alg. 1 driver are worth noting. (a) Argmax ties are broken by adding zero-mean Gaussian noise of scale 10^{-9} to the readout before the argmax, seeded per run for determinism. (b) On DM Math, any intermediate

integer whose absolute value exceeds $10^{12} - 1$ is saturated to that bound before re-tokenization, matching the representation range of §4.3; the truncation is silent by design, so a chain that overflows this range at any node cannot recover it. (c) The Mambino-Residual multiplication driver strips signs and short-circuits any operand-zero node symbolically at the composer level, calling the reducer only on positive-magnitude products and re-applying the sign product after; this partitions the primitive between the reducer (positive magnitude MUL) and the composer (sign rule, zero rule) but does not change what the reducer must learn.

H ROOT-OPERATOR LABEL-PRIOR BASELINE ON LONG-FORM LRA-LISTOPS

Miralles-González et al. (2025) observe that a substantial share of LRA end-to-end performance is driven by short-range statistics, and that on ListOps the label distribution conditional on the root operator is skewed: `MIN` favors 0, `MAX` favors 9, `MED` favors 4. This raises a natural baseline for the long-form setting: how much accuracy is recoverable from the root operator alone, absent any tree evaluation? On the 2000-example LRA-ListOps test partition, the deterministic root-conditional argmax (predict 0 if the root is `MIN`, 9 if `MAX`, 4 if `MED`, and the most-likely residue class if `SM`, estimated from the 96,000 training expressions (Feng, 2024)) reaches 0.371 ($742/2000$; alias `eval-A37z`, and the data-driven root-conditional argmax estimated on the same training set selects the identical rule), well below both the tree-composition band (Table 1) and the end-to-end SSM band. Our composer accuracies of 0.8735 (Alg. 1) and 0.9535 (Alg. 2) therefore sit well above what any label-prior mechanism recovers, as do the tree-composition baselines. The per-operator per-node accuracy of Mambino (Table 4) shows the reducer's accuracy is roughly uniform across `MIN`/`MAX`/`MED`, ruling out a label-prior explanation of the overall number.

I ALGORITHM-BY-TASK COVERAGE GRID

Table 3 organizes every parse-then-compose configuration reported in this paper as a grid of composition algorithm (hard-argmax vs closed-form PMF) crossed with reducer×task. Algorithm 2 is filled for every cell where the reducer emits a genuine probability distribution: LRA-ListOps at 0.9535 (Equations (7)–(10)); Mambino single-op cells at depth 1, which trivially equal Algorithm 1; Mambino `add_sub_multiple` chains at 0.986 (Equations (13)–(14) plus mixed-sign); and Mambino synthetic MUL-only chains at 0.622 (Equations (16)–(17)).

Table 3: All parse-then-compose configurations evaluated in this paper, organized by inference algorithm. EM_0 unless otherwise stated. Mambino-Residual's Alg. 2 cells are N/A because its readout is a continuous per-decade mean, not a categorical softmax.

Reducer (#params)	Task	Alg. 1	Alg. 2
Mambino, LRA-ListOps (110,192)	LRA-ListOps	0.8735	**0.9535**
Mambino, DM Math (515,623)	`add_or_sub`	0.973	0.973 (= Alg. 1 at depth 1)
Mambino, DM Math (515,623)	`add_sub_multiple`	0.989	0.986
Mambino, DM Math (515,623)	`mul` (depth 1)	0.917	0.917 (= Alg. 1 at depth 1)
Mambino, DM Math (515,623)	synth MUL-only chains d2–5	0.0015 ♭	0.622
Mambino-Residual (323,609)	`arithmetic.mul`	**1.000**	N/A (continuous-μ)
Mambino-Residual (323,609)	synth MUL-only chains d2–5	**0.9095**	N/A (continuous-μ)

♭ The DM Math Mambino under Algorithm 1 on synthetic MUL-only chains performs at noise floor because intermediate products of a chain leave the 1–4-digit operand range of its training calls (Appendix G); Mambino-Residual, shown two rows below, decodes them symbolically (Appendix D).

J LRA-LISTOPS PER-NODE ABLATION, FULL BREAKDOWN

Table 4 gives the full slice-level breakdown of the LRA-ListOps ablation summarized in §6.1. Every internal node in the 2000 LRA-ListOps test trees is scored against the subtree's ground-truth value (from the deterministic parser); "Total wrong" counts nodes where argmax disagrees with truth, and the $|pred - gt|$ breakdown decomposes those errors into off-by-1 (largely recoverable by `MIN`/`MAX`/`MED` at the parent) and off-by-≥ 2 (catastrophic across all four operators). Rows in bold are the largest slice-level effects, carrying the mechanism argument of §6.1.

K ABLATION NUMERIC TABLES

The three tables below carry the exact per-row numbers for the ε-injection ablation of §6 and Figure 4. All values in both tables are byte-anchored to the release artifacts named in Appendix L (Table 9).

Table 4: LRA-ListOps per-node ablation, Algorithm 1.

Slice	Full	Predoff	Δ
Overall per-node	0.9178	**0.9401**	+0.0223
Per operator			
MIN	0.9672	0.9868	+0.0196
MAX	0.9778	0.9886	+0.0108
MED	0.9449	0.9318	−0.0131
SM	0.7817	**0.8534**	**+0.0717**
Per operand count N			
$N = 2$	0.6958	0.7275	+0.0317
$N = 3$	0.8287	**0.9207**	**+0.0920**
$N = 4$	0.9585	0.9690	+0.0105
$N \in [5, 10]$	∼0.96	∼0.97	∼+0.01
Per depth from root			
$d = 1$ (composer output)	0.8735	**0.9180**	+0.0445
$d = 9$ (near-atomic)	∼0.95	∼0.96	+0.0135
Total wrong (out of 294,843)	24,223	17,657	−6,566
off-by-1 share of wrong	32.1%	**69.0%**	+36.9 pp
off-by-≥ 2 share	67.9%	31.0%	−36.9 pp

Table 5: LRA-ListOps ablation of the reducer's ε-injection branch. "Curriculum test" is the atomic-reducer training pipeline's held-out split (individual operator calls uniform over $N \in \{2..10\}$). "Per-node on composer" is the per-atomic-call accuracy on the sub-expressions actually invoked by the composer on the LRA-ListOps test set (294,843 nodes across 2000 test trees). "Composer acc" is root-level correctness on the same test set.

Metric	Full	Predoff	Δ
Curriculum test acc	0.9992	0.9986	−0.0006
Per-node acc on composer (Alg 1)	0.9178	**0.9401**	+0.0223
Composer acc, Alg. 1 (hard-argmax)	0.8735	**0.9180**	+0.0445
Composer acc, Alg. 2 (closed-form PMF)	0.9535	**0.9695**	+0.0160

Table 6: DM Math ablation of the reducer's ε-injection branch. All numbers are exact-match at zero absolute error (EM_0). Same protocol as Table 2: $n = 1000$ per module, seed 0, best checkpoints for each variant. Full: Mambino 515,623 params. Predoff: 450,599 params.

Module	Full	Predoff	Δ (Full−Predoff)
Alg. 1: add_or_sub	0.973	0.934	+0.039
Alg. 1: add_sub_multiple	0.989	0.973	+0.016
Alg. 1: mul (depth 1)	**0.917**	0.820	**+0.097**
Alg. 2: add_sub_multiple	**0.986**	0.823	**+0.163**
Alg. 2: synth MUL chains, $d \in [2, 5]$	0.622	0.516	+0.106

Table 7: DM Math ablation crossed with the seen/unseen partition of Mambino's DM Math training set (3,043,216 tuples within $|a|, |b| \leq 999$). An independent deterministic draw of $n = 1000$ per module from the same stream as Tables 2 and 6 (pooled accuracies agree within 0.8 pp), seed 0, Alg. 1. EM_0 with Wilson 95% CIs (CIs dropped from print for width; see release artifacts eval-A31z, eval-A32z, eval-A33z for CIs and full transcripts). Δ = Full − Predoff.

Module	Overlap	n_s/n_u	Full seen/unseen	Predoff seen/unseen	Δ_{seen}	Δ_{unseen}
mul (d 1)	68.8%	688/312	0.9913/0.7532	0.9637/0.5032	+0.028	**+0.250**
add_sub_multiple	84.0%	840/160	**1.0000**/0.9313	0.9964/0.8187	+0.004	**+0.113**
add_or_sub	51.3%	513/487	0.9961/0.9322	0.9844/0.8809	+0.012	**+0.051**

Table 8: Released checkpoints referenced in the paper. Two Mambino reducers share the same block design (§3) and differ in depth, state width, readout, and training data (LRA-ListOps vs. DM Math); we disambiguate them in prose by the benchmark they were trained on when both are in play. The alias prefixes (`ckpt-list-*`, `ckpt-cat-*`, `ckpt-digit-*`) reflect the original naming and are retained unchanged for artifact stability.

Reducer (paper name)	Params	Alias	Task family	DOI
Mambino (LRA-ListOps)	110,192	`ckpt-list-mambino-v10`	LRA-ListOps atomic reducer	pending
Mambino (DM Math)	515,623	`ckpt-cat-mambino-v10`	DM Math ADD/SUB/MUL/DIV	pending
Mambino-Residual	323,609	`ckpt-digit-mambino-v2`	DM Math MUL μ-refiner	pending
Predoff (LRA-ListOps)	101,072	`ckpt-list-predoff-v10`	LRA ablation (ε-injection removed)	pending
Predoff (DM Math)	450,599	`ckpt-cat-predoff-v10`	DM Math ablation (ε-injection removed)	pending
Tree-LSTM	111,898	`ckpt-list-treelstm-v1`	LRA-ListOps cross-arch ablation (N-ary Tree-LSTM cell, Tai et al. (2015))	pending

L REPRODUCIBILITY CATALOG

Tables 8 and 9 document the released-checkpoint aliases and per-claim evidence aliases for every numerical value cited in the main body. All identifiers refer to artifacts in the release archive (checkpoints and evaluation results); alias-to-file mapping is maintained in the archive's `reproduction_manifest.md`. DOIs will be populated with the deposited-checkpoint DOI at time of paper acceptance.

Table 9: Reported numerical claims and their released-artifact aliases. Each row identifies the value, the reducer that produced it, and the alias of the archived evaluation result. Alias numbering is not contiguous; `eval-A06` and `eval-A19` are not part of this release.

Claim	Value	Reducer	Release alias
LRA composer, Alg. 1	0.8735	Mambino	`eval-A01`
LRA composer, Alg. 2	0.9535 (1907/2000)	Mambino	`eval-A02`
DM `add_or_sub`, Alg. 1	0.973	Mambino	`eval-A03`
DM `add_sub_multiple`, Alg. 1	0.989	Mambino	`eval-A04`
DM `mul` (d1), Alg. 1	0.917	Mambino	`eval-A05`
DM `add_sub_multiple`, Alg. 2	0.986 (986/1000)	Mambino	`eval-A07`
`arithmetic.mul` (d1)	**1.000**	Mambino-Residual	`eval-A08`
Synth MUL chains $d=2$–5, Alg. 1	**0.9095**	Mambino-Residual	`eval-A09`
Synth MUL chains $d=2$–5, Alg. 2	0.622	Mambino	`eval-A10`
LRA per-node full, Alg. 1	0.9178	Mambino	`eval-A11`
LRA per-node predoff, Alg. 1	0.9401	Predoff	`eval-A12`
LRA composer predoff, Alg. 1	0.9180	Predoff	`eval-A13`
LRA composer predoff, Alg. 2	0.9695	Predoff	`eval-A14`
LRA per-op summary (MIN/MAX/MED/SM)	(Fig. 4(a))	derived	`eval-A15`
DM `add_or_sub`, Alg. 1, predoff	(Table 6)	Predoff	`eval-A16`
DM `add_sub_multiple`, Alg. 1, predoff	(Table 6)	Predoff	`eval-A17`
DM `mul`, Alg. 1, predoff	(Table 6)	Predoff	`eval-A18`
DM `add_sub_multiple`, Alg. 2, predoff	(Table 6)	Predoff	`eval-A20`
DM synth_mul, Alg. 2, predoff	(Table 6)	Predoff	`eval-A21`
LRA composer Tree-LSTM, Alg. 1 (cross-arch)	0.7445	Tree-LSTM	`eval-A22`
LRA composer Tree-LSTM, Alg. 2 (cross-arch)	0.9260	Tree-LSTM	`eval-A23`
`arithmetic.mul` (d1), $\delta \equiv 0$	**1.000**	Mambino-Residual	`eval-A08z`
Synth MUL chains $d=2$–5, $\delta \equiv 0$	**1.000**	Mambino-Residual	`eval-A09z`
DM `mul` (d1), Sonnet zero-shot	1.000	Sonnet 4.5	`eval-A24z`
DM `mul` (d1), Sonnet raw (unfiltered population)	0.910	Sonnet 4.5	`eval-A25z`
DM `add_or_sub`, Sonnet zero-shot	1.000	Sonnet 4.5	`eval-A26z`
DM `add_or_sub`, Sonnet raw (unfiltered population)	0.985	Sonnet 4.5	`eval-A27z`
DM `add_sub_multiple`, Sonnet zero-shot	0.445	Sonnet 4.5	`eval-A28z`
DM `add_sub_multiple`, Sonnet raw (unfiltered population)	0.455	Sonnet 4.5	`eval-A29z`
LRA composer, Sonnet zero-shot (n=100)	0.120	Sonnet 4.5	`eval-A30z`
DM `mul` (d1), Full×Predoff × Seen×Unseen 2×2	(four cells)	Mambino / Predoff	`eval-A31z`
DM `add_or_sub`, Full×Predoff × Seen×Unseen 2×2	(four cells)	Mambino / Predoff	`eval-A32z`
DM `add_sub_multiple`, Full×Predoff × Seen×Unseen 2×2	(four cells)	Mambino / Predoff	`eval-A33z`
LRA atomic-call overlap, per-N (Table 4)	per-N dict	Mambino	`eval-A34z`
LRA per-N × seen/unseen × {Full, Predoff} atomic-call accuracy (Wilson 95% CIs)	6-way table	Mambino / Predoff	`eval-A35z`
Carry-free decode of `naive_mul` (Lean 4 kernel)	`eval_conv` ⊢	(deterministic)	`proof-mul-carryfree`
Closed-form self-test (one-hot, ADD/MUL/signed ADD/SUB/MUL)	5000/5000	(deterministic)	`eval-A36z`
LRA root-operator label-prior baseline (Appendix H)	0.371	(deterministic)	`eval-A37z`