

Beyond Append-Only Decoding: A Recursive Edit-Based Encoder for Arithmetic Length Generalization

Bong Gyun Kang¹, Junyong Ahn¹, Jieun Byeon¹, Sungroh Yoon^{1,2†}

¹Interdisciplinary Program in Artificial Intelligence, Seoul National University, Seoul, South Korea,

²Department of Electrical and Computer Engineering, Seoul National University, Seoul, South Korea,
{luckypanda, tld215, bje5774, sryoon}@snu.ac.kr

Abstract

Algorithmic reasoning—internalizing a procedure from small examples and applying it at arbitrary scale—is a hallmark of human intelligence: a child who learns arithmetic on a few digits can, given time and paper, evaluate expressions of unbounded length. Modern language models cannot: Qwen3.5-27B, despite 100k-token reasoning contexts, degrades to below 10% accuracy on 30-digit multiplication. Departing from reasoning-through-scale, we introduce **RACE** (Recursive Arithmetic Control Encoder), a 3.6M-parameter Transformer trained from scratch that generalizes far better by recursively decomposing problems while keeping its working memory bounded. Two mechanisms drive this: token-level editing, which erases intermediate scratchwork, and self-calling, which recurses on sub-problems. Within a single model, RACE improves multiplication length generalization 4.7 times over the Arithmetic Transformer and, for the first time, achieves length generalization on subtraction and division ($\times 47.1$ and $\times 17.6$), while also solving nested, multi-operator expressions. Code is available at https://github.com/Pusheen-cat/RACE_2026.

1 Introduction

Large language models have advanced remarkably on competition-level mathematics, yet a curious gap persists at the most elementary level. Much of human intelligence is thought to rest on inductive inference: from a few examples, people grasp an underlying rule and apply it to unseen, larger instances (Arbib, 2012). Arithmetic reflects this cleanly: its rules are finite and learned from small cases, yet in principle extend to inputs of any length, with a clear notion of correctness at each step (Liu et al., 2023). It is thus among the purest tests of whether a system has acquired a

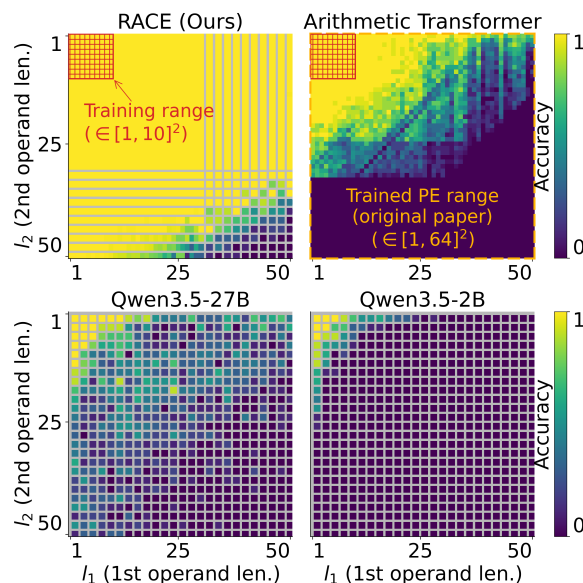


Figure 1: Comparison on the **multiplication** task between RACE (Ours; 3.6M), Arithmetic Transformer (ATF; 63.2M), and Qwen3.5-27B, Qwen3.5-2B. Both the ATF and our model were trained up to 10-by-10 length settings and used the same training samples.

genuine procedure rather than memorized a distribution. Yet modern neural networks fare surprisingly poorly—reliable at small sizes but brittle once operands grow beyond modest lengths, precisely the regime where rule-following should make length irrelevant (Deletang et al., 2023). Whether they can acquire such length-generalizing procedural knowledge remains a fundamental open question, with arithmetic as its sharpest testbed.

The reason this gap persists is structural. Whether a from-scratch arithmetic learner or a frontier reasoning model, nearly every approach shares one choice: an autoregressive decoder that generates left to right and is fundamentally append-only. Such a model solves a hard instance by writing an ever-longer chain of thought (CoT); DeepSeek-R1 (Guo et al., 2025), for example, carries out the naive quadratic schoolbook algorithm for multipli-

[†]Corresponding author.

cation, emitting an $O(N^2)$ trace for two N -digit numbers (Bavandpour et al., 2025). Every intermediate digit and partial product stays in the context and only accumulates, so two failures compound: the model must attend over a context far longer than anything seen in training, and most of that context is scratchwork acting as noise rather than signal. Long CoT is thus not merely expensive—it is the mechanism by which length generalization breaks down. Even Qwen3.5-27B-thinking (Qwen Team, 2026), a strong recent reasoning model, fails to reliably add, subtract, multiply, and divide once operands reach a few tens of digits, echoing a growing body of work on the same brittleness (Bavandpour et al., 2025). Mitigations that periodically clean the context (Yang et al., 2025; Lee and Kim, 2023) help, but remain bolt-ons over a decoder that cannot natively erase what it no longer needs.

Rather than scaling pre-trained models, a complementary line trains small networks from scratch on arithmetic, asking whether they learn the supervised algorithm and extrapolate beyond their training length. This program spans engineered position encodings, reversed-digit formats, scratchpads, and looped Transformers (LTF; Fan et al., 2025). Trained on up to ten 10-digit operands for addition and 10×10 multiplication, the most recent model extrapolates to roughly thirty 30-digit operands and 20×15 multiplication (Cho et al., 2025). Yet each result handles a single fixed task family, with encodings re-derived per operation—and as append-only decoders, these models share the same growing-context limitation. To our knowledge, no from-scratch model has shown length generalization on subtraction, division, mixed four-operation expressions, or nested parentheses—the operations children learn together as one system.

We answer with RACE (Recursive Arithmetic Control Encoder), a small encoder-based model trained from scratch that abandons the append-only decoder. RACE can operate directly on its input sequence—the very tokens it is given. Unlike an autoregressive model that only appends tokens to a growing context, RACE repeatedly rewrites the sequence in place, inserting, replacing, or deleting tokens at any position in a single forward pass. Solving a complete problem requires a sequence of such editing steps. We call this iteratively edited input sequence a “token tape”. Deletion erases scratchwork once consumed, so the tape stays bounded rather than accumulating the full trace. Every token also carries learned control elements,

including a self-call bit that lets the model invoke itself on a smaller subproblem—turning a hard instance into a recursive decomposition: long multiplication into shifted single-digit products, division into repeated subtraction, nested expressions into innermost-subexpression rewrites. Finally, a Global-Local positional encoding keeps long-range attention content-based, so it stays in-distribution far beyond the training length. Together these replace the long, noisy chain of thought that breaks autoregressive models with a short, controlled, recursive trace—shifting the design from memorization capacity toward control over computation.

Trained by supervised imitation on traces with operands of up to 10 digits, RACE achieves strong length extrapolation across all four operations and arbitrarily nested expressions. It improves multiplication length generalization 4.7 times (area) over the Arithmetic Transformer (ATF; Cho et al., 2025; Figure 1) and, for the first time in the from-scratch literature, achieves length generalization on subtraction and division ($\times 47.1$ and $\times 17.6$ of the training area)—for which no prior result exists. The generalization is not unlimited, with accuracy declining on the longest inputs, yet with only 3.6M parameters, the model surpasses both a Qwen3.5-27B-thinking LLM and the 63.2M ATF across every operation—evidence that, in this setting, the computational substrate and execution control are important determinants of length generalization.

Our contributions are as follows:

(i) We introduce a token-level edit mechanism augmented with a self-call operation. This allows recursive problem decomposition while bounding the working context, overcoming the scaling bottleneck of standard CoT.

(ii) We propose GLoPE, a Global-Local positional encoding that confines rotary phases to local windows while reserving content-only channels for long-range dependencies, enabling robust out-of-distribution attention at extrapolated lengths.

(iii) Our 3.6M RACE substantially outperforms the 63.2M ATF and a 27B Qwen3.5-thinking, marking the first from-scratch success on division, subtraction, and mixed-operator expressions.

2 Related Works

Why scaling alone does not fix arithmetic. Enlarging an autoregressive model does not buy length generalization. Dziri et al. (2023) show Transformers solve compositional tasks by lin-

Method	Operations covered	Len. gen.	Bounded memory	Native edit	Self call	Task-agnostic PE	Multi. gen. area
Pos. Coupling (2024)	copy/reverse, +, $N \times 2$ -digit	✓	✗	✗	✗	✗	$\times 2.3$
Abacus (2024)	sort, +, $N \times M$	✓	✗	✗	✗	✗	$\times 1.1$
PENCIL (2025)	3-SAT, QBF, Zebra	○	✓	○	✓	—	—
Looped TF (2025)	parity, copy, +, $N \times 2$ -digit (binary)	○	○	✗	○	—	$\times 1.5$
Arithmetic TF (2025)	+, $N \times M$	✓	✗	✗	✗	✗	$\times 8.9$
Qwen3.5 (2026)	+, −, $\times$, $\div$ (general purpose)	—	✗	✗	✗	—	—
RACE(Ours)	copy, +, −, $\times$, $\div$, sequential, nested	✓	✓	✓	✓	✓	$\times 41.9$

Table 1: Comparison of our model with representative from-scratch and scaled approaches. **Symbols:** ✓ = holds; ○ = holds partially or in a restricted setting; ✗ = no; “—” = not applicable. *Operations:* demonstrated tasks. *Len. gen.:* Solves test inputs longer than those seen in training. *Bounded memory:* the context does not grow with the full trace. *Native edit:* Supports native insertion/deletion. *Self-call:* Can recurse on a subproblem. *Task-agnostic PE:* One positional encoding serves all operations, with no per-task position IDs. *Multi. gen. (area):* multiplication length-generalization factor by area (product of operand lengths) of the largest reliably solved test regime over the training regime. Per-cell justifications appear in Appendix A.3.

earized subgraph matching, with accuracy decaying as depth grows, and Mirzadeh et al. (2025) find frontier models brittle to irrelevant perturbations of the same problem. Even the latest reasoning LLMs run the naive quadratic algorithm for multiplication (Bavandpour et al., 2025), so the trace fills the working window with scratchwork-noise as the problem grows (Yang et al., 2025). The bottleneck therefore cannot be explained by parameter count alone: the append-only substrate and its unbounded trace are important obstacles, motivating a change of substrate and a line that trains *small* Transformers from scratch to isolate what makes arithmetic length-generalize (Table 1).

Position encodings as the central lever. The most studied control in this line is the position encoding (PE), since the failure mode is positional: a model that knows the rule still cannot align digits of equal significance once an operand grows. NoPE decoders often extrapolate better than learned or relative variants (Kazemnejad et al., 2023); FIRE interpolates relative positions to extend the usable range (Li et al., 2024); and Randomized PE decorrelates train and test offsets (Ruoss et al., 2023). The decisive step is to inject *task structure*: Abacus tags each digit by its position within a number (McLeish et al., 2024), and Position Coupling shares a position ID across digits of equal significance (Cho et al., 2024), together pushing two-operand addition generalization (Sabbaghi et al., 2024; Zhou et al., 2024). Our GLoPE keeps this goal but drops the per-task IDs: it reserves content-only (NoPE) channels for long-range attention and confines rotary phases to a short local window, keeping out-of-distribution attention in-distribution.

Bounded memory and edit-based generation.

A complementary response bounds the working memory and makes editing native. Recursion of Thought and Re-Tuning re-invoke the decoder on each subproblem in a fresh context (Lee and Kim, 2023; Pasewark et al., 2024), and PENCIL instead erases a subcomputation in place once it returns (Yang et al., 2025). All rely on special tokens, with an inherent limitation: a left-to-right decoder cannot revise an emitted token, so edit locations must be fixed in advance rather than chosen when needed. Discrete-diffusion language models (DDLMS) move past this one-directional constraint, revising any position, and the Levenshtein Transformer and continuous-time edit flows make insertion and deletion atomic (Gu et al., 2019; Havasi et al., 2025; Nie et al., 2025; Ye et al., 2024, 2025). This is the right operation for arithmetic, where an algorithm is a sequence of rewrites: $(3+4) \times 5 \rightarrow 7 \times 5 \rightarrow 35$ is something an append-only decoder cannot do.

From single-operand addition to multiplication.

Existing from-scratch methods reach length generalization through task-specific machinery that does not transfer across operations: our principal baseline, the ATF, re-derives a position-coupling scheme and scratchpad per operation, and looped depth-adaptive models apply only to RASP-L-expressible tasks (Fan et al., 2025). As a result, *no from-scratch model length-generalizes on subtraction, division, mixed-operator, or nested expressions*. Our model closes this gap with a single architecture and PE that length-generalizes across all of them. We defer a detailed discussion of related work to Appendix A.

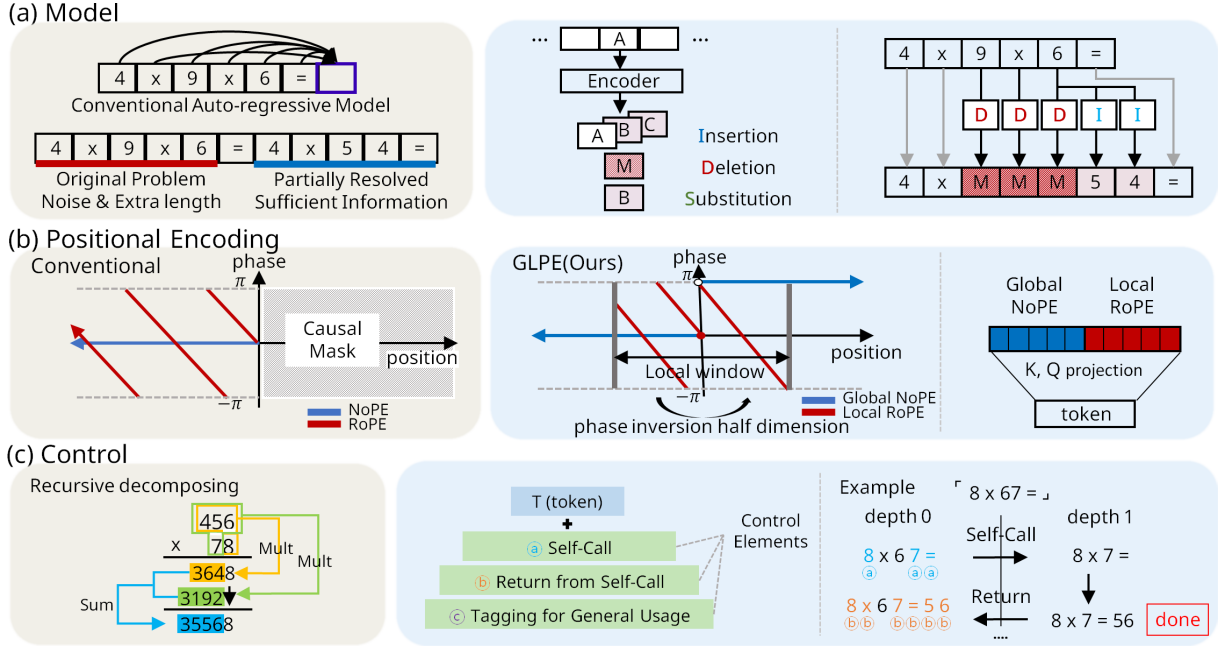


Figure 2: Method Overview. **(a) Model.** Unlike the autoregressive model, which appends to an ever-growing context that keeps the fully resolved problem in view as noise, our encoder *edits* its input sequence (the tape) in place, predicting insertion (I), deletion (D), and substitution (S) at every position in a single forward pass; deletion erases consumed scratchwork so the working tape stays bounded. **(b) Positional encoding.** GLoPE splits the head into a content-only global (NoPE) block and a window-confined local (RoPE) block, so long-range attention never leaves its training distribution. **(c) Control.** Each token carries control bits—(a) self-call, (b) return-from-self-call, (c) tag—that let the model recursively decompose a problem into subproblems and reassemble the partial results.

3 Method

Our model, RACE, departs from the append-only decoder along three axes (Figure 2): it *edits* a token tape in place rather than appending to it (§3.1); a Global-Local positional encoding (GLoPE) keeps long-range attention content-based, so it extrapolates far beyond the training length (§3.2); and per-token control bits let it recurse on subproblems (§3.3). RACE represents its input as a tape $X = (x_1, \dots, x_S)$ —the input sequence itself, rewritten rather than extended—where each cell $x_s = (\tau_s, c_s)$ pairs a vocabulary token τ_s with a control vector c_s . Two quantities organize the computation. A *step* is a single forward pass that rewrites the entire tape, the learned map $X \mapsto X'$; a problem is solved by a short sequence of such rewrites (e.g., $(3+4) \times 5 \rightarrow 7 \times 5 \rightarrow 35$). A *depth* is the current recursion level: a self-call (§3.3) pushes the active span to depth+1 and a return pops it to depth-1; at each step only cells at the current maximum depth are visible and editable.

3.1 Edit everywhere: insert, delete, substitute

One-shot multi-edit. A single step lets every position propose, in parallel, a local rewrite. The

encoder emits, at each position s , a block of $1 + M$ output slots, where M is the maximum number of insertable tokens per position. Output slot 0 rewrites τ_s *in place*: predicting the reserved MASK token *deletes* the position, predicting a different token *substitutes* it, and predicting τ_s leaves it unchanged. Slots $1 \dots M$ are new tokens *inserted* immediately after τ_s . A reconstruction step then assembles X' —splicing in the inserted tokens, dropping the MASK tokens, and repacking to a new length S' . The three classical edits of Figure 2(a) thus all happen at one editing step (one forward pass); a complete trajectory contains multiple editing steps. Deletion is the crucial edit operation: once a sub-result is consumed the model erases it, so the tape stays bounded instead of accumulating the full reasoning trace length (e.g. $O(N^2)$ for multiplication of size N ; Bavandpour et al., 2025) that breaks autoregressive decoders.

Parallel insertion. Rather than running a “decide-to-insert, then decide-what” loop, the model carries M MASK tokens alongside each input position and propagates them through the same blocks as the input tokens. Each MASK attends only to the input tokens and to itself—never to

other MASKs—so the M insertions at each position are *parallel*. Across all SM MASKs, the total insertion-stream attention cost is $O(S^2M)$ rather than $O(S^2M^2)$. At full utilization, where up to SM tokens are edited in parallel, this yields an amortized per-token editing cost as low as $O(S)$. This deliberately trades inter-insertion dependency for parallelism: coherence among simultaneously inserted tokens is carried entirely by the shared input-token context. Architecture details are in Appendix B.1.

3.2 GLoPE: global content, local phase

Why RoPE breaks at length. RoPE rotates each query/key pair at relative position $\Delta t = t_i - t_j$ by $\Delta t f_k$ for a decaying frequency ladder $\{f_k\}$, so the score depends on $\cos(\Delta t f_k)$ (Su et al., 2024). At test lengths many times the training length, the low-frequency channels reach phases never seen in training, the dot product drifts out of distribution, and accuracy collapses (Li et al., 2024). GLoPE removes long-range rotation entirely and confines it to a short window (Figure 2(b)).

Global content channel, local phase channel. We split the dimension d into a *global* block carrying no rotation and a *local* block carrying RoPE gated to a window $|\Delta t| \leq L$. The window is applied multiplicatively ($\{0, 1\}$) to the local block—rather than as an additive $-\infty$, which would suppress the entire score—so out-of-window pairs lose only their local phase while the global block keeps them connected:

$$s_{ij} = \frac{1}{\sqrt{d}} \left(\underbrace{q_i^g \cdot k_j^g}_{\text{global (NoPE)}} + \underbrace{\mathbf{1}[|t_i - t_j| \leq L] q_i^l \cdot k_j^l}_{\text{local (RoPE)}} \right), \quad (1)$$

with q^g, k^g and q^l, k^l the global and local sub-blocks of the query and key. The global block is the key device: carrying no notion of distance, its attention mechanism is identical at any sequence length, providing a long-range channel that never goes out of distribution while the distance-dependent channels stay confined to a window seen fully during training—which is what makes robust extrapolation possible. The exact head partition—the global NoPE region, a region encoding which insertion slot $i \in \{1, \dots, M\}$ a mask occupies, and two windowed local regions over t and the composite axis $t+i$ —with the window width L and frequency list, is given in Appendix B.2.

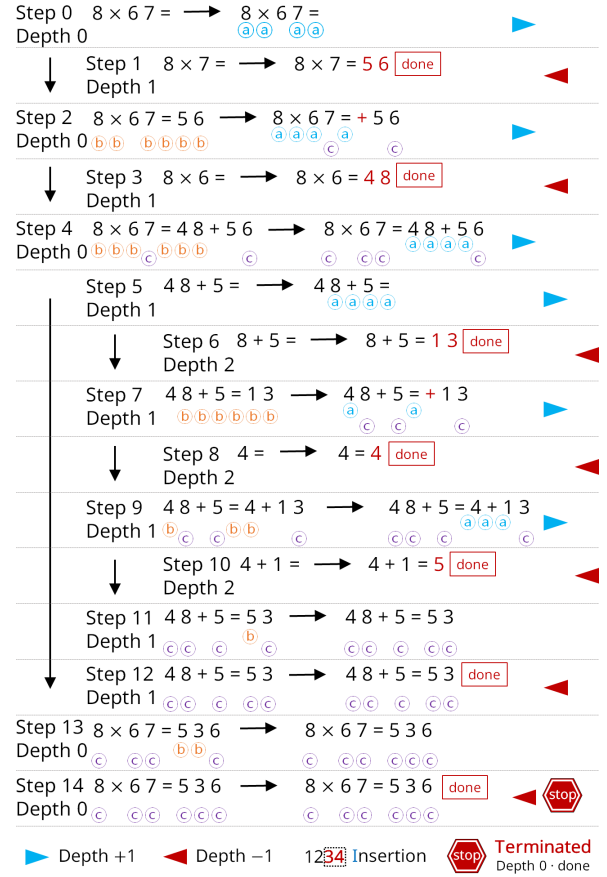


Figure 3: Full trajectory of multiplication example. Shows the evolution of tokens together with their control elements. Through a self-call, the model processes only a subset of tokens at a deeper depth and then returns the result to the parent level. Control elements are consistent throughout: (a) self-call, (b) return from self-call, and (c) tag.

Phase inversion for bidirectional attention. A bidirectional encoder has no causal mask. Without it, NoPE conveys no positional information at all, and RoPE—depending only on the relative offset $\Delta t = t_i - t_j$ —cannot tell “before” from “after.” We therefore invert the phase of half the dimensions in both the NoPE and RoPE blocks, breaking this left/right symmetry so the model can distinguish the two directions. More details of GLoPE are provided in Appendix B.2.

3.3 Recursion with control: self-call

Per-token control. Each cell carries a control vector $c_s = (c_s^{(a)}, c_s^{(b)}, c_s^{(c)}) \in \{0, 1\}^3$ of *independent* binary bits, so the model can combine roles within one token. The bits are embedded by a small learned matrix added to the token embedding and read back through a tied control head—a second, parallel “vocabulary” the model both consumes

Stage	Example ($\ell, n, d = 3$)	Training range / grid (cells)	samples/cell	Total samples
copy	123=	lengths 1–40	250k	10.0M
+ (two-operand)	123+456=	$(\ell_1, \ell_2) \in \{1..10\}^2$	100k	10.0M
$\times$ (two-operand)	123 $\times$ 456=	$(\ell_1, \ell_2) \in \{1..10\}^2$	10k	1.0M
– (two-operand)	123–456=	$(\ell_1, \ell_2) \in \{1..10\}^2$	10k	1.0M
$\div$ (two-operand)	123 $\div$ 456=	$\{1..10\}^2, \ell_2 \leq \ell_1 + 2$	10k	~ 0.7 M
seq. exp. $\{+, -, \times\}$	123–456 $\times$ 789=	$(n, max-d) \in \{2..10\} \times \{1..10\}$	10k	0.9M
nested seq. exp. $\{+, -, \times\}$	(123–456) $\times$ 789=	$(n, max-d) \in \{2..10\} \times \{1..10\}$	10k	0.9M

Table 2: Per-stage training configuration. Base LR is $2e-4$ for all tasks. For the sequence expression (seq. exp.) stages, n is the number of operands (2–10) and d the maximum operand length (1–10); each cell guarantees at least one operand at exactly d digits. “nested seq. exp.” places $n-2$ valid parenthesis pairs per expression.

and emits. The three bits play fixed roles (Figure 2(c), 3): (a) self-call, (b) return, (c) tag. Note that bit (b) is set deterministically by the engine, rather than by the model, when a subcall returns.

(a) Self-call grows the recursion. Every position with $c^{(a)} = 1$ seeds a deeper level: its depth is incremented, and on the next step *only* the cells at the current maximum depth are presented, while lower-depth cells are frozen and hidden. Marking a span as a self-call thus exports it as the input to a fresh subproblem (e.g. long multiplication into a shifted single-digit product). Because the parent context is hidden during the subcall, the working window is the subproblem alone.

(b) Return marks the result. When the model emits *done* at the current depth, one level is popped: the just-finished cells have their depth decremented and their return bit $c^{(b)} = 1$ set before being folded back into the parent. The returning value thus arrives flagged as a just-finished subtree, and the model attends to that flag to choose its next rewrite (e.g. summing partial products).

(c) Tag. The third bit is a free-form meta-channel with no fixed semantics: the model may annotate a token with a signal useful to its own policy, carrying state along the tape.

Depth-local termination. *done* is depth-local: it ends only the current level, and the episode terminates only when *done* fires *and* depth is already 0. This rule turns the editor into a recursive solver: it descends through self-calls, solves each base case, and unwinds through returns over one bounded tape. Figure 3 traces 8×67 : at stage 0 (depth-0) the model self-calls the subproblem 8×7 ; at depth-1 it computes $8 \times 7 = 56$ and signals *done*, and the result returns flagged with $c^{(b)}$. The waiting depth-0 tokens then resume: recognizing from $c^{(b)}$ that 8×7 is done, the model self-calls the next

subproblem 8×6 , and tags the ones digit of 56 with $c^{(c)}$ —setting it aside to add to the subsequent product—while emitting a $+$ token.

4 Result

Training setup. We continually train a *single* model across *all* operations and expression types using supervised imitation from teacher trajectories restricted to short sequence lengths. The curriculum progressively moves from simpler to more complex operations while evaluation measures generalization beyond the training regime. One set of 3.6M weights is carried through the entire curriculum (Table 2) in dependency order $\text{copy} \rightarrow + \rightarrow \times \rightarrow - \rightarrow \div \rightarrow \text{seq.exp.} \rightarrow \text{nested seq.exp.}$. The “Training range / grid (cells)” column specifies the range of operand lengths used during training. After each stage, we evaluated the model’s generalization on the corresponding task.

We match the number of training samples with the LTF (Fan et al., 2025) and ATF (Cho et al., 2025) baselines on the operations they support, with the same training range. Crucially, our step-level supervision is the same kind that these from-scratch models already require (e.g., ATF trains on hand-designed scratchpad traces): our teacher trajectory is the edit-substrate analogue of these task-specific scratchpads, specifying intermediate rewrites rather than an appended CoT. This comparison therefore tests computational substrates under comparable procedural supervision: ATF also imitates hand-designed, operation-specific scratchpads. RACE does not autonomously discover the arithmetic procedures; it learns their execution from short teacher traces and is evaluated on whether that execution composes and extrapolates to unseen lengths. Qwen3.5 is not a directly controlled baseline, but a reference for the scaled append-only paradigm; its thinking-mode prompting and evalu-

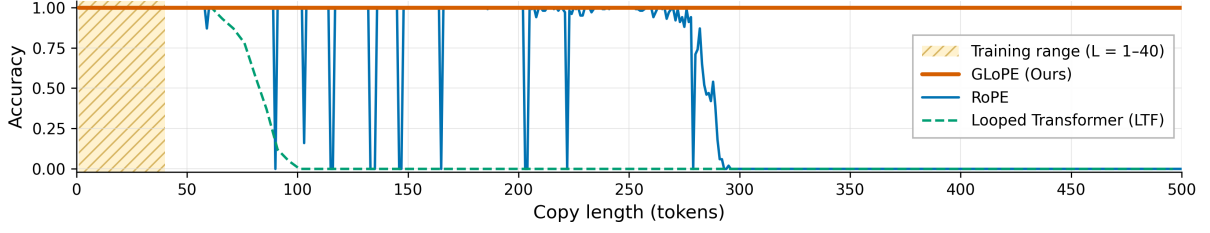


Figure 4: This is a copy task in which the model receives an input such as “123=” (length 3) and generates “123=123”. The model was trained on sequences up to length 40 and evaluated on inference lengths up to 500. Accuracy was measured using 100 random samples for each length.

ation protocol are documented in Appendix D.4.

For the subtraction and division tasks, which we present for the first time, subtraction targets include negative values, while division requires generating results up to three decimal places. All results are evaluated using “exact match”. Full hyperparameters, architecture, data generation, and the evaluation protocol are in Appendix C.

4.1 Copy task

The copy task simply requires reproducing a given numeric sequence, teaching the model that both sides of the equality sign should be identical. Figure 4 compares our model using GLoPE and RoPE, along with LTF. Only our model with GLoPE maintained perfect generalization up to length 500, exceeding the training length by more than $12\times$, demonstrating the limitation of RoPE.

4.2 Addition and Multiplication

We compare against ATF on addition and multiplication. We define length generalization as extrapolation to sequence lengths absent from training (OOD). ATF does not satisfy this setting, as its learned per-digit positional embeddings require fixing a maximum inference length and training on positions up to that bound. Consequently, its evaluation is effectively in-distribution. In contrast, our model assumes no predefined length limit, enabling genuine OOD generalization far beyond the training range. For the addition and multiplication tasks, we performed evaluation on inputs up to a 100×100 grid.

Addition. We kept the ATF methodology unchanged, but only changed the task from adding multiple numbers of the same length to adding two numbers of different lengths. Figure 5 shows that RACE stays at 100% exact-match past 70×70 , degrading only on the longest inputs. ATF, with its maximum position fixed at 40, holds up to that

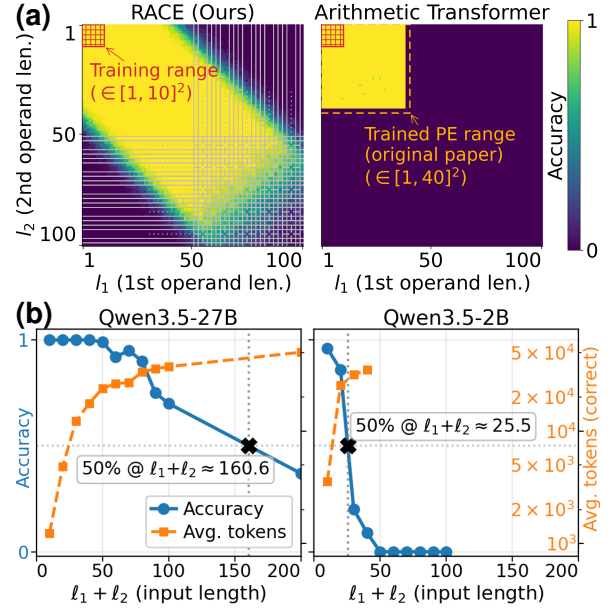


Figure 5: Comparison on the **addition** task among RACE (Ours), ATF, and Qwen3.5-27B, Qwen3.5-2B. Both the Arithmetic Transformer and our model were trained up to 10-by-10 length settings and used the same number of training samples.

bound and collapses the moment either operand exceeds it. Raising the bound to an unrealistic 110 does not help: accuracy degrades starting at length 20, and length generalization fails (Appendix E.1). Notably, our model achieves a generalization area of $\times 61.4$, compared to $\times 14.4$ for ATF, representing more than a fourfold improvement.

Multiplication. Figure 1 compares multiplication performance across 4 models. RACE generalizes multiplication to an area $41.9\times$ larger than the 10-by-10 training space and remains accurate even on multiplication tasks requiring the generation of 70-digit outputs. This substantially exceeds the $8.9\times$ generalization area achieved by ATF and extends nearly an order of magnitude beyond the multiplication range handled by a recent LLM; Qwen3.5-27B-thinking.

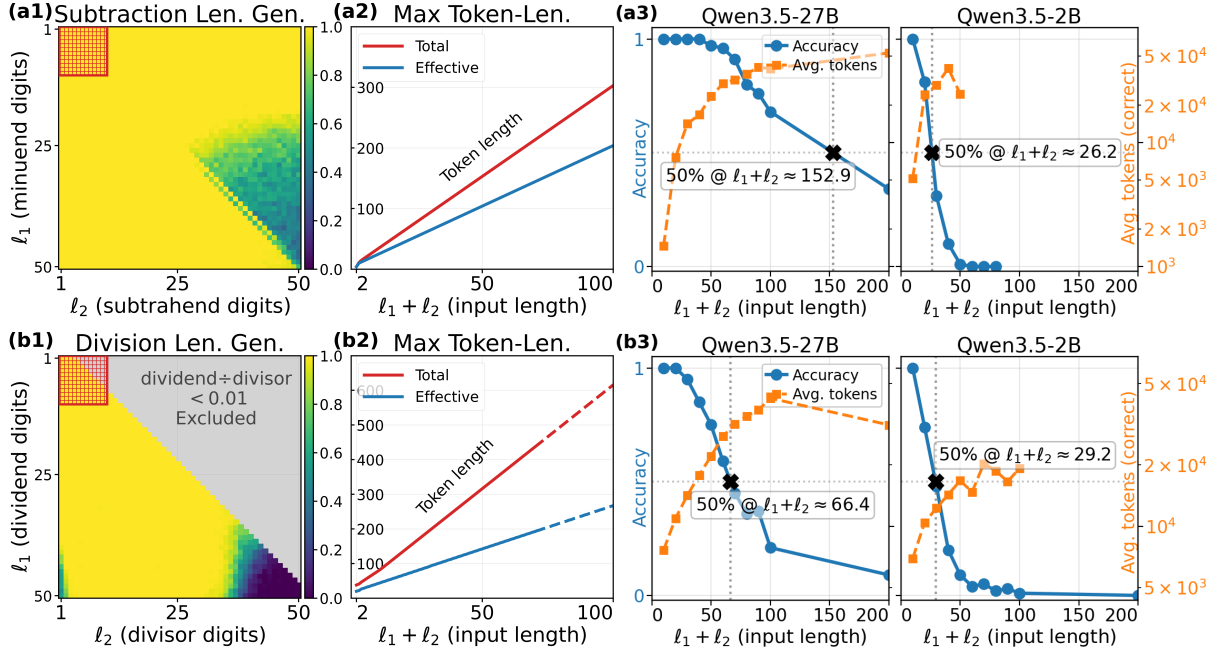


Figure 6: Top: (a1) Training and inference accuracy on the subtraction task. (a2) The maximum number of total tokens the model holds versus the effective tokens it actually processes over the entire course of subtraction. (a3) Average performance as a function of input length and the number of tokens used for inference on the Qwen3.5-27B and Qwen3.5-2B thinking-mode reference models. Bottom: The division task is shown in the same order; for division, evaluation is performed only on combinations yielding results of at least 0.01.

4.3 Frontiers: Subtraction, Division, Nested Sequence Expression

To our knowledge, we are the first to extend generalization to the full set of arithmetic operations, encompassing subtraction and division. These operations are challenging because, unlike addition and multiplication—which operate solely on positive integers—they require extending the operational framework to negative numbers and rational numbers (i.e., decimal values). Division, in particular, inherently involves a trial-and-error process of determining how many times the divisor fits into the dividend. Even on these more demanding tasks, RACE achieves robust generalization through learning. These results provide an initial from-scratch benchmark for subtraction and division.

Subtraction and Division. Figure 6 demonstrates that RACE executes the subtraction and division procedures at greater lengths to solve challenging problems. RACE achieves a generalization space that is $47.1\times$ wider for subtraction and $17.6\times$ wider for division, and in both cases the maximum number of input tokens required throughout the computation scales linearly with the input length (i.e., a substantially smaller memory). Compared

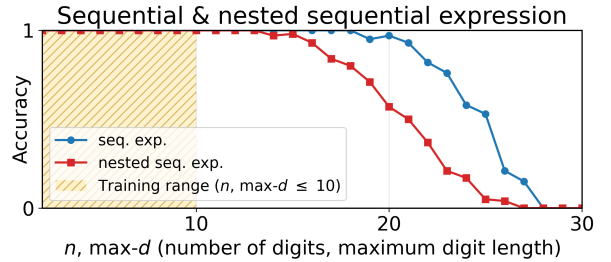


Figure 7: Seq. exp. generalization experiments over varying lengths (n operands, max-d length) and various operator types ($+$, $-$, $\times$). To reduce computational cost, we evaluate the cases where $n = \text{max-d}$. (Note that the numbers need not have equal lengths.)

against the performance of the Qwen3.5-thinking, RACE overwhelmingly outperforms the 2B model and attains performance comparable to that of the 27B model.

Sequential and Nested Sequential Expressions. RACE goes one step further, achieving generalization to sequential arithmetic problems that alter the flow of computation through diverse operators and parentheses. Figure 7 shows that both sequential expressions (seq. exp.) and nested sequential expressions (nested seq. exp.) attain $1.5\text{--}2\times$ length generalization in terms of both the length of the numbers and the number of operators.

Ablation	Copy	Addition	Multiplication
No token erasure (trace accumulates)	$\times 12.5^*$	$\times 36.93$	$\times 6.56$
GLoPE $\rightarrow$ RoPE	$\times 6.75$	$\times 3.23$	$\times 1.35$
GLoPE $\rightarrow$ NoPE	$\times 8.60$	$\times 20.99$	$\times 6.83$
GLoPE without phase inversion	$\times 12.5^*$	$\times 51.96$	$\times 6.04$
Without control bit (c), tag	$\times 12.5^*$	$\times 5.76$	$\times 2.80$
Without control bit (b), return	$\times 2.74$	$\times 17.70$	$\times 3.54$
RACE (full)	$\times 12.5^*$	$\times 61.4$	$\times 41.9$

Table 3: Component ablations. Values are length-generalization area relative to the training region. *The copy evaluation ceiling is $\times 12.5$ (perfect accuracy through length 500). Detailed interpretations are in Appendix D.1.

Continual Learning Analysis. Training the operations continually, warm-starting each stage from the previous one, converges faster and more stably than training from scratch, suggesting that earlier skills transfer to later ones (Appendix E.2).

4.4 Component ablations and scope analyses

Table 3 isolates token erasure, GLoPE, and recursive control under the same training and evaluation protocol as the full model. Removing erasure most strongly harms multiplication, whose trajectories are longest. Replacing GLoPE with RoPE or NoPE, removing phase inversion, or removing either the tag or return signal all reduce extrapolation, with the largest gaps on multiplication. Self-call bit (a) cannot be removed while retaining the recursive teacher transitions.

Two further analyses delimit the claim. A separate neural module learns the engine’s fixed recursion-state transition map and reaches the evaluation ceilings on held-out lengths, but it has not yet replaced the engine in the full solver (Appendix D.2). Increasing the arithmetic base from 10 to 200 enlarges the primitive fact set and degrades multiplication, showing that capacity can become a limiting factor when the rule-set grows (Appendix D.3).

5 Conclusion

We introduced RACE, a small from-scratch encoder that replaces the append-only decoder with an edit-and-recurse substrate: token-level edits erase consumed scratchwork to keep the tape bounded, a self-call operation turns hard instances into recursive subproblems, and a Global–Local positional encoding keeps long-range attention in-distribution far beyond the training length. With only 3.6M parameters, a single model length-generalizes across all four operations and nested multi-operator expressions, surpassing the con-

trolled 63.2M ATF baseline and the 27B Qwen3.5 thinking-mode reference. These results show that, in this arithmetic length-generalization task, the computational substrate and execution control strongly affect length generalization, whereas parameter scaling does not account for the observed improvement.

Limitations

Our results are obtained with a single small encoder (3.6M parameters) trained from scratch on synthetic arithmetic tasks. We have not scaled RACE to, or integrated it into, a general-purpose language model, nor evaluated it on natural-language inputs or tasks beyond arithmetic. Our conclusions are therefore deliberately narrow: in the supervised arithmetic settings studied here, the computational substrate and execution control are important determinants of length generalization, while model capacity can become limiting as the set of primitive facts grows. Whether the same edit-and-recurse substrate provides comparable benefits at language-model scale, or interacts favorably with broadly acquired linguistic competence, remains an open question.

Our supervised-imitation curriculum also relies on expert teacher trajectories. Although this requirement is comparable to the step-level supervision used by the from-scratch baselines, such as ATF’s per-operation scratchpads, suitable trajectories are readily available for arithmetic but much harder to obtain for open-ended reasoning tasks. Extending RACE beyond arithmetic may therefore require weaker supervision or trajectories generated through other means. In addition, the observed length generalization is strong but not unlimited: accuracy declines on the longest inputs. Thus, although the policy learned from teacher traces extrapolates beyond the training lengths, it does not behave as a perfect algorithm executor.

One potential direction is to investigate RACE in conjunction with large pretrained discrete-diffusion language models (DDLMS), whose bidirectional and non-autoregressive operation provides a natural setting for in-place editing. Future work should determine whether deletion and recursive self-calls can be incorporated effectively at this scale and whether the resulting computational control transfers to broader reasoning tasks.

Acknowledgements

This research was supported by a grant of the MD-PhD/Medical Scientist Training Program through the Korea Health Industry Development Institute (KHIDI), funded by the Ministry of Health & Welfare, Republic of Korea, Institute of Information & Communications Technology Planning & Evaluation (IITP) grants funded by the Korea government (MSIT) (No. RS-2021-II211343, Artificial Intelligence Graduate School Program (Seoul National University)), and NO. RS-2022-II220959), National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2022R1A3B1077720), the BK21 FOUR Program of the Education and Research Program for Future ICT Pioneers, Seoul National University in 2026, Mobile eXperience (MX) Business, Samsung Electronics Co., Ltd, NAVER Digital Bio Innovation Research Fund, funded by NAVER Corporation (Grant No. [3720260110]), and NVIDIA Academic Grant Program.

References

- Michael A Arbib. 2012. *Brains, machines, and mathematics*. Springer Science & Business Media.
- Alireza Amiri Bavandpour, Xinting Huang, Mark Rojin, and Michael Hahn. 2025. [Lower bounds for chain-of-thought reasoning in hard-attention transformers](#). In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 3274–3306. PMLR.
- Hanseul Cho, Jaeyoung Cha, Pranjal Awasthi, Srinadh Bhojanapalli, Anupam Gupta, and Chulhee Yun. 2024. [Position coupling: Improving length generalization of arithmetic transformers using task structure](#). In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37.
- Hanseul Cho, Jaeyoung Cha, Srinadh Bhojanapalli, and Chulhee Yun. 2025. [Arithmetic transformers can length-generalize in both operand length and count](#). In *International Conference on Learning Representations (ICLR)*.
- Gregoire Deletang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. 2023. Neural networks and the chomsky hierarchy. In *The Eleventh International Conference on Learning Representations (ICLR)*.
- Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaïd Harchaoui, and Yejin Choi. 2023. [Faith and fate: Limits of transformers on compositionality](#). In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36.
- Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. 2025. Looped transformers for length generalization. In *International Conference on Learning Representations (ICLR)*, volume 2025, pages 14502–14520.
- Jiatao Gu, Changhan Wang, and Junbo Zhao. 2019. [Levenshtein transformer](#). In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. [DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning](#). *Nature*, 645(8081):633–638.
- Marton Havasi, Brian Karrer, Itai Gat, and Ricky T. Q. Chen. 2025. [Edit flows: Variable length discrete flow matching with sequence-level edit operations](#). In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 139037–139070.
- Bong Gyun Kang, HyunGi Kim, Dahuin Jung, and Sungroh Yoon. 2023. [Clear: Continual learning on algorithmic reasoning for human-like intelligence](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 48768–48780.
- Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. 2023. [The impact of positional encoding on length generalization in transformers](#). In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36.
- Soochan Lee and Gunhee Kim. 2023. [Recursion of thought: A divide-and-conquer approach to multi-context reasoning with language models](#). In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 623–658. Association for Computational Linguistics.
- Shanda Li, Chong You, Guru Guruganesh, Joshua Ainslie, Santiago Ontanon, Manzil Zaheer, Sumit

- Sanghai, Yiming Yang, Sanjiv Kumar, and Srinadh Bhojanapalli. 2024. [Functional interpolation for relative positions improves long context transformers](#). In *International Conference on Learning Representations (ICLR)*.
- Bingbin Liu, Jordan T Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. 2023. Transformers learn shortcuts to automata. In *The Eleventh International Conference on Learning Representations (ICLR)*.
- Sean McLeish, Arpit Bansal, Alex Stein, Neel Jain, John Kirchenbauer, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, Jonas Geiping, Avi Schwarzschild, and Tom Goldstein. 2024. Transformers can do arithmetic with the right embeddings. *Advances in Neural Information Processing Systems*, 37:108012–108041.
- Iman Mirzadeh, Keivan Alizadeh-Vahid, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2025. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *International Conference on Learning Representations (ICLR)*, volume 2025, pages 94743–94765.
- Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2025. Large language diffusion models. *Advances in Neural Information Processing Systems*, 38:50608–50646.
- Eric Pasewark, Kyle Montgomery, Kefei Duan, Dawn Song, and Chenguang Wang. 2024. [Re-tuning: Overcoming the compositionality limits of large language models with recursive tuning](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10422–10437. Association for Computational Linguistics.
- Qwen Team. 2026. [Qwen3.5: Towards native multi-modal agents](#).
- Anian Ruoss, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Róbert Csordás, Mehdi Bannani, Shane Legg, and Joel Veness. 2023. [Randomized positional encodings boost length generalization of transformers](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1889–1903. Association for Computational Linguistics.
- Mahdi Sabbaghi, George Pappas, Hamed Hassani, and Surbhi Goel. 2024. [Explicitly encoding structural symmetry is key to length generalization in arithmetic tasks](#). *arXiv preprint arXiv:2406.01895*.
- Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.
- Chenxiao Yang, Nathan Srebro, David Mcallester, and Zhiyuan Li. 2025. [PENCIL: Long thoughts with short memory](#). In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 71092–71156. PMLR.
- Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. 2025. [Beyond autoregression: Discrete diffusion for complex reasoning and planning](#). In *International Conference on Learning Representations (ICLR)*.
- Jiacheng Ye, Shansan Gong, Liheng Chen, Lin Zheng, Jiahui Gao, Han Shi, Chuan Wu, Xin Jiang, Zhenguo Li, Wei Bi, and Lingpeng Kong. 2024. [Diffusion of thought: Chain-of-thought reasoning in diffusion language models](#). In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Joshua Susskind, Samy Bengio, and Preetum Nakkiran. 2024. [What algorithms can transformers learn? a study in length generalization](#). In *International Conference on Learning Representations (ICLR)*.

A Supplementary related works

A.1 Theoretical learnability of arithmetic

A parallel line of theory asks when length generalization is possible in principle. [Zhou et al. \(2024\)](#) propose the RASP-Generalization Conjecture: a decoder-only Transformer tends to length-generalize on a task when the true next-token function can be written as a short RASP-L program that works at all input lengths, a criterion that captures many known successes and failures (addition and parity, which lack simple RASP-L solutions, are among the hard cases). Constructively, [Cho et al. \(2024\)](#) prove a 1-layer existence result for two-operand addition, and [Cho et al. \(2025\)](#) extend the construction to multi-operand addition with embedding dimension logarithmic in operand length and count. These results concern representability and idealized constructions; they do not characterize the behavior of practical decimal models, reinforcing our reading that empirical studies must examine computational substrate and execution control alongside representational capacity.

A.2 Discrete diffusion and reasoning

Our edit-based substrate is closely related to the discrete-diffusion language-model line. At scale, [Nie et al. \(2025\)](#) train an 8B masked diffusion language model (LLaDA) from scratch that is competitive with a similarly sized autoregressive LLM

in in-context learning and notably mitigates the reversal curse, evidence that core LLM capabilities need not be tied to left-to-right decoding. On reasoning specifically, Diffusion-of-Thought (Ye et al., 2024) imports chain-of-thought into diffusion language models and reports a small diffusion model outperforming a much larger autoregressive one on multi-digit multiplication, Boolean logic, and grade-school math, while exhibiting self-correction. Ye et al. (2025) push this further with Multi-Granularity Diffusion Modeling, reaching 91.5% and 100% on Countdown and Sudoku versus 45.8% and 20.7% for autoregressive baselines, and explain the gap through subgoal imbalance—diffusion learns difficult subgoals that elude autoregressive training. These models, however, retain a fixed-length, mask-based denoising procedure that does not natively change sequence length during generation; the deletion and self-call operations in our model are precisely what add bounded, variable-length editing on top of this substrate.

A.3 Justification for Table 1

We compare against five from-scratch or scaled baselines along six axes: the *operations* each method demonstrably handles; whether it *length-generalizes* (Len. gen.); whether its *working memory is bounded*, i.e. the context the model must attend to does not grow with the full execution trace; whether *insertion/deletion are native* model operations rather than appended trigger tokens (Native edit); whether the method can *recursively call itself* on a subproblem (Self-call); and whether a *single position encoding serves all operations* without per-task position IDs (Task-agnostic PE). We use ✓/✗ for a clear yes/no, ◯ for partial or restricted support, and “—” when an axis does not apply to the method’s design. Below we justify each row from the corresponding paper. All claims are taken from the cited works.

Abacus (McLeish et al., 2024). Abacus introduces a learned positional embedding that tags each digit by its position within a number. Trained on additions of up to 20 digits, it generalizes to 100+ digits (a $6\times$ extrapolation, versus the prior $2.5\times$ state of the art), and the authors further report length generalization on multiplication (up to 15-digit operands) and on sorting; hence ✓ for length generalization and an operations entry of $+$, $\times$ (with sorting). The model is a standard decoder-only Transformer, optionally augmented with input injection

and recurrent (looped) blocks; neither erases or compresses the running context, so working memory is *not* bounded (✗), and there is no token-level insertion/deletion (✗) or self-call (✗). The encoding is *task-specific*: it requires reversing the digits of each number and indexes positions *within* a numeric span rather than over the raw sequence, a construction tailored to arithmetic and not transferable as-is to other tasks (✗ for task-agnostic PE).

Position Coupling (Cho et al., 2024). Position Coupling assigns the *same* position ID to digits of equal significance. A small Transformer trained on 1–30-digit additions generalizes to 200-digit additions ($6.67\times$); the paper additionally demonstrates the method on $N \times 2$ multiplication (and on multi-summand addition, copy/reverse, and a 2D task), so the operations entry is $+$, $N \times 2$ and length generalization is ✓. The model performs ordinary next-token prediction with a standard decoder, so working memory is unbounded (✗), with no native editing (✗) and no self-call (✗). Crucially, the coupling rule must be *hand-designed per task*: the authors state as an explicit limitation that designing the position coupling for tasks whose structure is implicit remains future work. The position encoding therefore encodes task structure and is not task-agnostic (✗).

Arithmetic Transformer (Cho et al., 2025). This is our principal from-scratch baseline. It combines task-specific scratchpads with multi-level Position Coupling (a bi-level scheme for multi-operand addition and a tri-level scheme for multiplication) and achieves roughly $2\text{--}3\times$ length generalization on multi-operand addition and on $N \times M$ multiplication—hence operations $+$, $N \times M$ and ✓ for length generalization. This scratchpad is itself a hand-designed, per-operation, step-by-step trace that the model is trained to imitate—the same form of step-level supervision as the teacher trajectories used here, differing only in that it is appended autoregressively rather than applied as in-place edits. We mark working memory as *not* bounded (✗): although the scratchpad lets the model “focus on a fixed number of tokens per next-token-prediction step,” this describes a bounded *per-step attention span*, not a bounded working context—the scratchpad is appended autoregressively and grows with the trace. Editing is not native (the scratchpad is append-only, ✗) and there is no recursive self-call (the scratchpad is a flat multi-step expansion, ✗). Because both the scratchpad format and the cou-

pling level are re-derived for each operation, the position encoding is not task-agnostic (✗).

Looped Transformer (Fan et al., 2025). Looped Transformers reuse a single decoder block for an adaptive number of steps and length-generalize on tasks with a known iterative (n -RASP-L) solution. The evaluated tasks include parity, copy, addition, and multiplication (the paper reports that looped training improves length generalization over vanilla next-token prediction on addition, copy, and multiplication); the precise base/operand setting of the multiplication task is not central to our comparison, so we list the representative operations as “parity, copy, +, $\times$.” We mark length generalization as $\circ$ because the guarantee is restricted to n -RASP-L-expressible tasks and the training signal includes the required number of steps; it does not extend to general decimal expressions. Working memory is $\circ$: the method uses end-to-end supervision with *no* intermediate scratchpad, looping over a latent state, so the context does not grow with the trace—though there is no explicit reduction mechanism either, only the absence of growth. There is no token-level insertion/deletion (✗). We mark self-call $\circ$ with a caveat: the model adapts its *number of loop iterations* to the input length, which is iteration rather than true recursive self-invocation on a subproblem; there is no explicit self-call primitive. Finally, the method’s contribution is the looping mechanism, not a position encoding, so the task-agnostic-PE axis does not apply (“—”).

PENCIL (Yang et al., 2025). PENCIL augments autoregressive generation with a reduction rule that erases intermediate thoughts once a sub-computation returns. It is evaluated on three hard combinatorial-reasoning tasks—3-SAT (NP-complete), QBF (PSPACE-complete), and the Zebra (Einstein) logic puzzle—rather than on arithmetic length-generalization benchmarks; hence the operations entry “3-SAT, QBF, Zebra” and $\circ$ for length generalization (it shows robustness to growing *problem size* on these tasks, but does not target arithmetic operand-length extrapolation). Working memory is bounded (✓): the reduction rule keeps the maximum context at the space complexity of the computation (e.g. reducing Einstein’s puzzle from 151,192 to 3,335 tokens) and provably simulates a Turing machine in $O(T)$ tokens with $O(S)$ context. We mark native editing $\circ$: erasure is real, but it is implemented as a pattern-matched [CALL]/[SEP]/[RETURN] *reduction rule* on top of

an append-only decoder (when the generated sequence matches $C\ T\ A$, it is rewritten to $C\ A$), not as free insertion/replacement/deletion at arbitrary positions. Self-call is supported (✓): [CALL] initiates a subroutine whose details are later discarded, supporting recursive, nested-function-like structure. PENCIL does not center on a position encoding, so the task-agnostic-PE axis does not apply (“—”).

B Detailed Method

This appendix gives the full specification of the components summarized in the main text: the edit architecture and its sampling rule, the attention mechanism (parallel-insertion dependencies and directional scoring), the GLoPE region partition, and the engine-side control semantics. Throughout, the model edits a tape $X = (x_1, \dots, x_S)$ with cells $x_s = (\tau_s, c_s) \in \mathcal{V} \times \{0, 1\}^3$; each step is a single forward pass of a bidirectional encoder that emits, per position s , $1 + M$ output slots ($M = \text{max_insertion}$) together with the corresponding control logits and a global *done* logit.

B.1 Edit architecture

Output slots and reconstruction. For each visible position s the head produces vocabulary logits of shape $(S, 1+M, |\mathcal{V}|)$ and matching control logits. Slot 0 is the *replacement* for s : vocabulary id 0 = MASK is interpreted as “delete this position,” any other id $\neq \tau_s$ is a substitution, and τ_s itself is the identity. Slots $1 \dots M$ are candidate tokens to be inserted immediately after s . A reconstruction step materializes the successor tape X' by splicing the predicted slots into the old tape, dropping all MASK and PAD cells, and repacking the result to length S' . The model is therefore free to delete, replace, and grow the sequence at any position on every step.

Pre-inserted masks. Rather than autoregressively deciding *whether* to insert and then *what* to insert, every position permanently carries M mask tokens. Concretely, each position has a real hidden stream $h_s \in \mathbb{R}^d$ and a parallel insertion stream $g_{s,1}, \dots, g_{s,M} \in \mathbb{R}^d$, the latter initialized to mask embeddings and propagated through the same Transformer blocks. At sampling time we keep only the prefix of insertion slots up to and including the first emitted MASK: a MASK in an insertion slot signals “no further insertion here,” so all later slots at the same position are discarded. Slot 0

is exempt from this rule, since a MASK there denotes a genuine deletion rather than a stop signal.

Insertion-stream dependencies. Letting every slot attend to every other slot would cost $O((SM)^2)$. We instead restrict the insertion stream to two destinations. Each mask attends with full cross-attention to all real keys (*insert*→*main*), and to itself through a single extra score $q^{\text{ins}}_{s,m}! \cdot k^{\text{ins}}_{s,m}$ (*insert*→*self*); masks never attend to other masks at the same position (*insert*↔*insert*). Hence each mask’s output $g_{s,m}$ is a function of its own embedding and the real-token context only. With SM mask queries, each attending to S real-token keys, the resulting total insertion-stream cost is $O(S^2M)$ rather than $O((SM)^2)$.

Two consequences follow. (i) The M insertions are genuinely parallel: no slot’s prediction depends on another’s content, so no left-to-right structure leaks across slots and any coherence among simultaneously inserted tokens must come from the shared real-token context. (ii) A single forward pass can produce up to SM token edits in parallel. At this maximum utilization, amortizing the total $O(S^2M)$ computation over SM token edits yields a per-token editing cost as low as $O(S)$.

Directional attention. Because the model reads the tape bidirectionally it cannot use a causal mask, yet it must still distinguish “before” from “after.” Each head is split into a front half and a back half; both compute scores, the back-half score is multiplied by a sign mask ϵ_{ij} , and the two are summed before the softmax:

$$A_{ij} = \text{softmax}_j \left(\frac{1}{\sqrt{d}} (q_i^f \cdot k_j^f + \epsilon_{ij} q_i^b \cdot k_j^b) \right), \quad (2)$$

where $\epsilon_{ij} = +1$ for $j \leq i$ and $\epsilon_{ij} = -1$ for $j > i$. This yields a content-based front/back asymmetry without any positional mask, and the same sign flip is applied to the summed GLoPE score below.

Termination and heads. The *done* flag is read at the EOS position of slot 0 as a single scalar $\ell_{\text{done}} = \ell_{\text{DONE}} - \ell_{\text{EOS}}$, with $\text{done} \sim \text{Bernoulli}(\sigma(\ell_{\text{done}}))$. The LM head is tied to the token embedding. The model exposes only (vocab logits, control logits, ℓ_{done}) per slot/position.

B.2 GLoPE region partition

Within each attention direction the head dimension is partitioned into four regions. Write t for tape

position, $i \in \{0, \dots, M\}$ for insertion-slot index, and $D_I \equiv \text{dim_insert}$ for the per-head budget allotted to insertion geometry. Table 4 lists the regions; the “rotation” columns name the axis each region rotates with RoPE (“—” is NoPE).

Reg.	Width / dir	t -rot.	i -rot.	Window
A	$\frac{d}{2} - \frac{3}{4}D_I$	—	—	—
B	$\frac{1}{4}D_I$	—	i	—
C	$\frac{1}{4}D_I$	t	—	$ \Delta t \leq L$
D	$\frac{1}{4}D_I$	$t+i$	$t+i$	$ \Delta t \leq L$

Table 4: GLoPE head partition (per direction), with window $L = \max(4, M+1)$ and doubling period list $\{1, 2, 4, \dots, 8L\}$ for the local channels.

The roles are:

- **A — global NoPE.** No rotation and no window. This is the long-range channel: since distance never enters it, its attention behavior is identical at any sequence length, so there is no out-of-distribution positional phase to drift into at extrapolated lengths.
- **B — NoPE on t , RoPE on i .** Encodes *which* insertion slot $(1, \dots, M)$ a mask occupies.
- **C — local RoPE on t , windowed.** Precise short-range token-distance information.
- **D — local RoPE on the composite axis $t+i$, windowed.** Because B and C already give independent handles on Δi and Δt , the diagonal collision in D is intentional: it acts as a single “tape position” coordinate shared by real and inserted tokens. The period-1 entry rotates by 2π per step (identity for integer offsets) and is retained as a content-only channel inside the local window.

Score split. Let D_g be the combined width of A+B per direction; q^g, k^g denote the first D_g channels (global) and q^l, k^l the remainder (local). The score sums a global term and a window-gated local term,

$$s_{ij} = \frac{1}{\sqrt{d}} \left(q_i^g \cdot k_j^g + \mathbf{1}[|t_i - t_j| \leq L] q_i^l \cdot k_j^l \right), \quad (3)$$

followed by the directional sign flip of Eq. (2). The window enters *multiplicatively* (a $\{0, 1\}$ gate), not as an additive $-\infty$, so tokens outside the window still attend through the global channels and only lose the local contribution. The same split is applied to *insert*→*main* cross-attention; *insert*→*self*

stays unsplit because its offset $|\Delta t| = 0$ is always inside the window. For the production configuration ($d = 384$, 4 heads $\Rightarrow$ head dim 96, $D_I = 48$) each region is 12 dims per direction and the period list $\{1, 2, 4, 8, 16, 32\}$ matches the six local pairs exactly.

B.3 Control-flow engine

The control vector $c_s \in \{0, 1\}^3$ is embedded by a learned matrix $\mathbf{E}_c$ and added to the token embedding as $c_s^\top \mathbf{E}_c$; the output control logits are recovered through the transpose of the same matrix, so the control “vocabulary” is tied across input and output exactly like the token LM head. Each bit is an independent Bernoulli, allowing arbitrary combinations within one token.

Self-call (bit a). Every position with $c^{(a)} = 1$ has its depth incremented, after which its control bits are cleared. On the next forward pass only the cells at the current maximum depth are presented to the model; all lower-depth cells are frozen and hidden until the subcall returns. The tokens at the marked positions are exported as the input of the subproblem, which the environment can verify as a structurally valid expression.

Return (bit b). When *done* fires at the current depth, the engine finds the highest-depth cells, decrements their depth by one, and sets their return bit $c^{(b)} = 1$ before folding them back into the parent tape. Bit b is thus *engine-set*, not model-emitted: it signals to the resuming parent that a subtree just finished and carries its produced value. Training masks b out of the loss and inference clamps it to 0, preventing spurious model emission. We additionally penalizes emission of SOS/PAD/EOS/DONE at non-terminal positions to keep special-token usage clean (deletion via MASK is a legitimate edit and is not penalized).

Tag (bit c). Bit c has no engine-imposed semantics; the model is free to use it as a meta-channel. For example, the model may use it to mark a token as finalized, indicating in subsequent steps that the token has completed its role in the computation and no longer requires editing. This lets the model carry discrete auxiliary signals along the tape without enlarging the vocabulary.

Depth-local termination. The *done* flag ends only the current depth. With $d_{\max}$ the current maximum depth, the episode terminates iff $\text{done} \wedge (d_{\max} = 0)$; otherwise the engine pops one level

and the parent resumes. This is what makes the model a recursive solver rather than a one-shot generator: it descends through self-calls, solves base cases, and unwinds through returns over one bounded tape, never expanding a nested derivation into a flat string.

C Training and evaluation details

Objective and curriculum. The model is trained to reproduce the teacher action at each step by per-step negative log-likelihood; the only loss term is this likelihood (§C.2). The seven stages of Table 2 are run sequentially, each warm-started from the converged checkpoint of the previous one. The *copy* stage teaches the tape mechanics (insertion, deletion, and the control channel) on the identity map; the four two-operand stages teach addition, multiplication, subtraction, and division on operands of up to ten digits; and the final two stages teach evaluation of sequential multi-operator expressions, first without and then with nested parentheses. All operands are base-10, and an operand’s length is its number of decimal digits.

C.1 Architecture

The model is the bidirectional edit encoder of §3.1–§3.3 with 2-layer Transformer blocks, hidden width 384, 4 attention heads (head dimension 96), and forward+backward (directional) attention combined per block. Each active position carries up to $M = 3$ parallel insertion slots, a control embedding of width 24 for the three control bits, and the GLoPE head partition of §3.2. The vocabulary has 26 symbols (Table 5; the ten digits, operators, and a small set of teacher mid-trajectory markers) with reserved ids MASK = 0, SOS = 1, PAD = 2, EOS = 3, DONE = 4, and the control head has 3 bits, matching the teacher. The maximum tape length is 3000. Dropout is 0.1, and the model has $\approx 3.6\text{M}$ trainable parameters.

C.2 Optimization and loss

We use AdamW with gradient clipping at $\|g\| \leq 1.0$, in bf16 autocast (no gradient scaler; optimizer state in fp32). The base learning rate is $2\text{e-}4$ for all stages, with a linear warmup over the first 1% of estimated steps from 0 to the base LR, followed by a half-cosine decay to a floor of $0.01 \times$ the base LR. At each step the model predicts an element distribution and a done distribution; the per-step

Id	Token	Role
0	[MASK]	deletion / “no token here”
1	[SOS]	start-of-sequence sentinel
2	[PAD]	padding
3	[EOS]	end-of-sequence sentinel
4	[DONE]	done sentinel [†]
5	=	equation marker
6–9	+ − × ÷	arithmetic operators
10–19	0–9	decimal digits
20	.	decimal point (division)
21–22	()	parentheses (nested expressions)
23	%	division remainder marker
24	–	operand-bracket marker (division)
25	b	borrow marker (subtraction)

Table 5: Vocabulary (26 ids). Tokens 23–25 are teacher scratchpad markers used internally during a trajectory. [†]The binary done signal is the [DONE] logit minus the [EOS] logit.

objective is

$$\mathcal{L} = -\mathbb{E}[(elem_logp \cdot ctl_keep \cdot final_mask) + done_logp]$$

where *final_mask* zeros insertion positions at or after the first predicted MASK, and *ctl_keep* zeros the engine-set *result* control bit so the model is never asked to predict it (the result bit is set by the return mechanism, not emitted by the policy). No auxiliary regularization is used.

C.3 Curriculum and data

Each stage streams an iterable dataset of teacher trajectories: tasks are sampled from a fixed budget, expanded on the fly into per-step tuples, and shuffled through a per-worker buffer of 4000 tuples (16 workers) so that consecutive minibatch items rarely come from the same trajectory. Global batch size is 1000–2000 depending on the VRAM of the GPUs used. Table 2 summarizes the per-stage configuration.

The four two-operand stages use the operand-length grid $\{1, \dots, 10\}^2$ directly. For division the grid is restricted to $\ell_2 \leq \ell_1 + 2$, the set of operand-length pairs for which the teacher’s quotient-magnitude filter ($|a/b| \geq 10^{-2}$) admits a valid problem, leaving ~ 0.7 M tasks. The sequential stages use a factored sampler over (number-of-operands, maximum-operand-length): operators are drawn uniformly from $\{+, -, \times\}$ per slot, and the nested stage inserts $n-2$ syntactically valid parenthesis pairs. Tasks for which the teacher fails to produce a terminating trajectory are dropped from the stream.

For a controlled comparison with the Arithmetic Transformer (ATF) (Cho et al., 2025), we sepa-

ately retrained ATF using the *same* task-specific sample budgets as our model: 100k samples per length cell (10M total) for addition and 10k per cell (1M total) for multiplication. Under these matched budgets, our model matches or exceeds ATF. Across our training stages, the supervised loss is already near zero after roughly 0.1M samples, so subsequent samples chiefly serve to confirm convergence rather than drive it.

C.4 Compute and convergence

Training uses 4-way DDP (NCCL) on a single node of four 48 GB GPUs in bf16. Each stage of the curriculum completes within 24 hours of GPU wall-clock time, and the supervised loss converges to the numerical noise floor ($\leq 10^{-3}$) well within that budget on every stage.

C.5 Evaluation protocol

All evaluation uses greedy decoding: argmax over the vocabulary at each position, a $p > 0.5$ threshold for the control and done bits, an over-length budget of $3 \times$ the initial input length for the working tape, and a per-run cap on the number of edit steps. Accuracy is exact-match on the final tape. We sweep the two-operand grid by (ℓ_1, ℓ_2) and the sequential grid by (number-of-operands, maximum-operand-length), recording per-cell accuracy over a fixed number of held-out instances. The inference forward pass runs in the same bf16 autocast as training.

D Analyses and evaluation details

D.1 Component ablations

The ablations in Table 3 use the same training data, curriculum stage, optimization, and evaluation grid as RACE; each variant changes only the named mechanism. The reported quantity is length-generalization area relative to the corresponding training region, so $\times 1$ denotes no extrapolation beyond the training grid.

Token erasure. In the no-erasure variant, intermediate states remain on the tape and the trace always accumulates. Copy, which requires little scratchwork, remains at the evaluation ceiling, whereas addition degrades moderately and multiplication drops from $\times 41.9$ to $\times 6.56$. The increasing gap with trace length is consistent with accumulated intermediate states hindering extrapolation.

GLoPE. Replacing GLoPE with either RoPE or NoPE substantially reduces generalization. RoPE

loses the length-invariant global channel, whereas NoPE loses precise local positional structure. Removing phase inversion preserves copy and much of addition but reduces multiplication to $\times 6.04$, indicating that left/right directional information becomes especially important for the longer multiplication trajectories.

Control signals. Removing the free tag bit (c) or the engine-set return bit (b) substantially reduces addition and multiplication extrapolation. Bit (a) is not directly ablatable: removing self-call eliminates the recursive operation and makes the corresponding teacher transitions undefined rather than yielding a comparable non-recursive variant. These results isolate the contribution of the signals used for execution-state management; they do not imply that RACE discovered those control semantics without procedural supervision.

D.2 Learning the engine’s recursion-state transitions

The engine refers to the rule-based processing performed outside the model. For example, upon a self-call, it increments the recursion depth, collects the tokens marked with control element (a), and passes the resulting active subsequence to the model for the next forward step. The engine has two conceptually distinct parts. Part A reconstructs and compacts the tape by applying predicted insertions, substitutions, deletions, and deterministic cleanup associated with control markers. Part B applies the fixed recursion-state transitions: depth updates, active-token visibility, and the call/return transformations. Neither part contains an arithmetic table or an operation-specific solver, but Part B is externally specified in the end-to-end system.

To test whether Part B can be internalized, we train a separate transition-prediction module on pre/post-transition state pairs extracted only from training-grid trajectories. It uses the same encoder architecture and GLoPE, except that the $M = \text{max_insertion}$ axis is flattened before input. Given tokens, control fields, the *done* action, and a per-token recursion-depth counter, the module predicts a same-length sequence containing the updated tokens, control fields, and terminated action. Evaluation measures exact match of the post-transition state, not end-to-end arithmetic accuracy.

The module learns the fixed Part B map and extrapolates across the evaluated lengths. This result does not establish that substituting the learned module for Part B inside the iterative inference

Transition model	Copy	Add.	Mult.
Learned Part B	$\times 12.5^*$	$\times 100^*$	$\times 100^*$

Table 6: Generalization factor for exact recursion-state transition prediction. *No degradation occurs within the evaluation budget; all three values are evaluation ceilings.

loop would preserve solver accuracy: distribution shift from its own transition errors could accumulate, and Part A remains deterministic. End-to-end internalization is therefore future work.

D.3 Rule-capacity scaling with base- N arithmetic

The main experiments use a small, homogeneous set of decimal rules and do not test a large collection of heterogeneous or conflicting procedures. As a controlled capacity stress test, we increase the number of primitive arithmetic facts while keeping the schoolbook procedures fixed. We train one model for each base $N \in \{10, 50, 100, 200\}$, representing every base- N digit as a distinct token; the number of possible single-digit input pairs therefore grows as N^2 . Addition and multiplication are trained jointly in one mixed stage with one million samples per task over operand lengths $\{1, \dots, 10\}^2$. Evaluation extends to 100×100 for addition and 30×30 for multiplication. These runs are comparable across bases but are not directly comparable to the curriculum-trained main results.

Performance declines as the base grows, particularly for multiplication. Moreover, the experiment scales a homogeneous fact set while holding the procedure fixed; handling heterogeneous, interacting, or conflicting rule systems remains open.

D.4 Qwen3.5 thinking-mode evaluation protocol

Qwen3.5 is included as a reference for the scaled append-only reasoning paradigm, not as a directly controlled baseline; ATF is the controlled from-scratch comparison. The 27B and 2B checkpoints were executed with `enable_thinking=True`, producing an explicit `<think>` reasoning block.

- **Prompting.** We use one fixed three-shot prompt per task (copy, addition, subtraction, multiplication, division, sequential expressions, and nested expressions), and reuse it for every problem within that task. Each exemplar has the form Question:

Metric	Base 10	Base 50	Base 100	Base 200
Addition, in-distribution accuracy	1.00	1.00	1.00	1.00
Addition, generalization area	$\times 95.1$	$\times 94.5$	$\times 48.8$	$\times 32.3$
Multiplication, in-distribution accuracy	1.00	1.00	0.997	0.705
Multiplication, generalization area	$\times 9.0^*$	$\times 8.1$	$\times 7.7$	$\times 2.2$

Table 7: Rule-capacity stress test. *The base-10 multiplication result reaches the 30×30 evaluation ceiling.

135+623= followed by `final_answer: 758` (addition example). The prompt ends with Your response must include: `final_answer: <answer>; division` additionally requests three decimal places. Generation is actively initiated inside an open `<think>` tag.

- **Reasoning budget and continuation.**

Chain-of-thought is allowed with an initial budget of up to 120,000 generated tokens. A run truncated inside reasoning receives one continuation pass with 10,000 additional tokens. If it remains inside the reasoning block, we append `</think>` and `final_answer:` to give the model an explicit final-answer opportunity.

- **Decoding.** We use temperature 1.0, top- p 0.95, top- k 20, and presence penalty 1.5, following the checkpoint’s recommended thinking settings. Tool calling is disabled, so no external calculator is available.

- **Problems and answer extraction.** Each task/length cell contains 100 problems generated from a deterministic seed. The parser takes the last `final_answer:` following the closed reasoning block and tolerates boxed answers, Markdown, thousands separators, and numerically equivalent forms such as $41.0 = 41$. To make the reference evaluation favorable, any numerical value in the reasoning block that matches the ground truth is also counted as correct. Division uses an absolute tolerance of 0.001. The complete prompts, continuation logic, seeds, and parser are released with the code.

E Detailed Results

E.1 Addition: comparison with the Arithmetic Transformer

What we are measuring. Our objective is out-of-distribution (OOD) length generalization: a model

trained on short operands should remain correct on operands far longer than any seen in training. This is the property the from-scratch literature aims for, and it is the property our evaluation is designed to expose. As we explain below, the Arithmetic Transformer (ATF) does not actually measure it—its “length generalization” is, by construction, in-distribution.

Task setting. ATF is reported on multi-operand addition where every operand has the *same* length. From the perspective of an arithmetic solver this is the easier regime: equal-length operands align digit-by-digit by construction, so the model never has to resolve offset alignment between operands of unequal size. To avoid giving our model an unfair advantage, we retrain ATF—leaving its shift-and-add scratchpad and training pipeline untouched—on the same two-operand, unequal-length addition we use ourselves, over $(\ell_1, \ell_2) \in \{1, \dots, 10\}^2$. Both models therefore see operands of up to ten digits during training under identical task distributions.

Why ATF’s length generalization is in-distribution by construction. Length generalization is meaningful only if the test-time length distribution is withheld during training; otherwise success at long inputs reflects memorized coverage rather than extrapolation. ATF cannot satisfy this. Its positional scheme is position coupling: a learned positional embedding is assigned per digit so that digits of equal significance across operands receive the same encoding (equivalently, a learned PE indexed by digit position). Because each position’s embedding must be *learned*, ATF has to assume, before training, the maximum operand length $P_{\max}$ it will see at inference, and it trains by shifting operands across every position up to $P_{\max}$ so that each embedding receives gradient. Two consequences follow. First, no input longer than $P_{\max}$ can be represented at test time, so the usable test length is capped at training time. Second, and more importantly, every position appearing at test time has already appeared during

training: the test-time positional distribution is a subset of the training one. What is reported as length generalization is therefore in-distribution interpolation within a pre-assumed range, not extrapolation to unseen lengths.

Our model makes no such commitment. GLoPE reserves content-only (NoPE) channels for long-range attention and confines rotary phase to a local window, so attention at positions far beyond the training length stays in-distribution *without any position-specific parameter being learned for those positions*. The lengths it is evaluated on are genuinely OOD—never represented, in any form, during training.

Results at the matched bound ($P_{\max} = 40$). We first set ATF’s maximum position to $P_{\max} = 40$, i.e. ℓ_1 and ℓ_2 may each range up to 40 digits, while both models are trained only on lengths up to 10. Figure 5 reports per-cell exact-match accuracy over the (ℓ_1, ℓ_2) grid. Our model remains at 100% accuracy well beyond 70×70 , degrading only on the longest inputs. ATF remains accurate up to its declared bound of 40 and then collapses abruptly once either operand exceeds it; the failure coincides exactly with the edge of the pre-declared position range, consistent with the analysis above.

Extending the bound does not help ($P_{\max} = 110$). A natural objection is that ATF could simply declare a larger bound. We therefore train a second ATF with $P_{\max} = 110$ —an unrealistic setting, since it presupposes knowledge of the test length and inflates the learned positional table accordingly (Figure 8). This does not recover length generalization. Accuracy instead begins to degrade from around operand length 20, with more failing cells as either operand grows, and never approaches the clean generalization of our model. Enlarging the position table spreads the same training budget over more learned embeddings without granting any of them the content-based, length-agnostic behavior GLoPE provides, so the underlying limitation persists.

E.2 Continual curriculum learning: the effect of warm-starting

Our curriculum trains a single set of weights through progressively harder operations, warm-starting each stage from the converged checkpoint of the previous one rather than reinitializing (copy $\rightarrow + \rightarrow \times \rightarrow - \rightarrow \div \rightarrow \text{seq.} \rightarrow \text{nested}$;

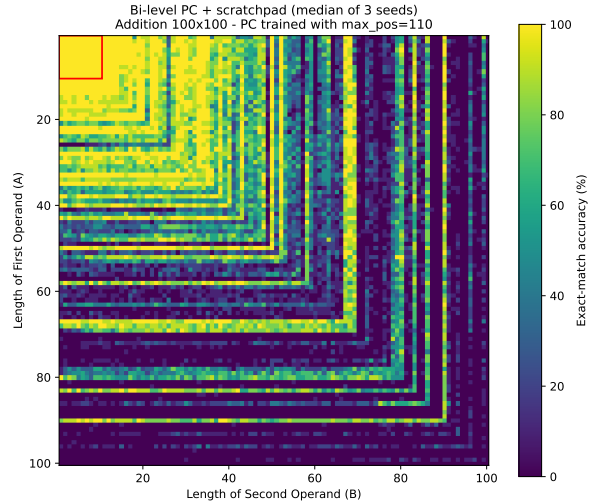


Figure 8: ATF when the training positional bound is extended to $P_{\max} = 110$.

§4). This mirrors the way humans acquire arithmetic, building each skill on the ones already learned (Kang et al., 2023). We test whether this staged warm-starting is in fact beneficial, as opposed to a convenience.

Concretely, we compare two ways of beginning the multiplication stage under an otherwise identical setting: warm-starting from the model that has already learned addition, versus training from a random initialization. Figure 9 reports the training loss for both. Warm-starting from the addition-trained model drives the loss down faster and converges more stably, whereas the randomly initialized run is slower and less stable.

This indicates that learning the operations in stages, rather than in isolation, is helpful: skills acquired on simpler operations transfer to harder ones, and continual training over the curriculum improves the model’s ability to acquire and execute the procedurally supervised policies.

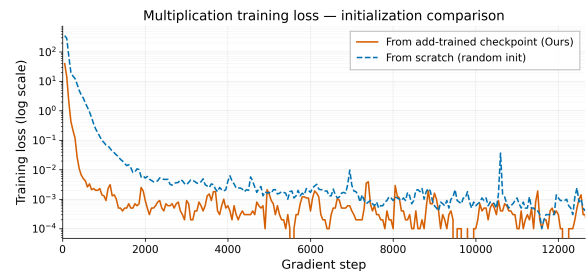


Figure 9: Training loss during multiplication learning with and without warm-starting from an addition-trained model. Warm-starting accelerates convergence and improves training stability.

F LLM Usage Statement

We used LLMs only for translation and language refinement during manuscript preparation and as assistive tools during code development. All experimental procedures and manuscript content were thoroughly reviewed and verified by the authors.