# TRAINING RECURSIVE LANGUAGE MODELS FOR HIERARCHICAL COMPOSITIONAL GENERALIZATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Language models may excel at applying familiar reasoning patterns yet struggle when those patterns must be combined in a novel way. Recursive inference offers a natural way to tackle novel compositions by decomposing them into potentially familiar subproblems and composing their results. Prior work has largely studied recursion for long-context processing and efficient inference, leaving its effect on compositional generalization unclear. We introduce a controlled Countdown benchmark for hierarchical compositional generalization and compare recursive and linearized execution on the same reasoning trajectories with matched training and inference token budgets. **Recursive inference improves accuracy on held-out compositions up to $\times 7$ while generating about 40% fewer tokens.** Separated subproblem contexts also permit training on reasoning trajectories whose total length greatly exceeds one context window, and extending search may further help by exploring alternative ways to combine these familiar constituents. Training on wider recursive trajectories and allowing wider search at inference increases compositional accuracy by $\times 2.8$, whereas doubling the generation limit yields only a 10% relative increase. Allocating search budgets across recursion levels improves the accuracy–compute trade-off, while mixing search budgets during training achieves similar quality with roughly half as many generated tokens. Finally, we build an adaptive curriculum by utilizing the compositional relationships between the tasks, doubling the held-out compositional accuracy.

## 1 INTRODUCTION

Many reasoning problems can be solved by combining familiar concepts rather than deriving every step from scratch. A mathematical proof can combine known lemmas in a new theorem (Saparov et al., 2023), and a program can nest familiar functions in an unseen call structure (Shi et al., 2024). In both cases, the components may be known, but their arrangement creates a new reasoning problem. For language models, this tests whether familiar patterns can be reused in a hierarchy absent from training, requiring compositional generalization (Fodor & Pylyshyn, 1988; Lake & Baroni, 2018; Keysers et al., 2020). Prior work has found that models often struggle with novel combinations of known reasoning steps (Press et al., 2023; Dziri et al., 2023; Zhao et al., 2024; Xiao & Zhao, 2025). Evaluating this ability requires separating changes in solution structure from changes in numerical values or problem length.

Recursive inference provides a natural way to execute hierarchical reasoning. A model can call itself on a subproblem, solve it in a separate context, and return the result to the parent problem. Prior work has used recursive calls to manage long contexts (Zhang et al., 2025), reduce latency (Lian et al., 2025), and support longer reasoning trajectories (Pan et al., 2025; Yang et al., 2026). Whether this inference format improves transfer of familiar reasoning patterns to unseen combinations remains unclear. We study that question by comparing recursive execution with a linearized version of the same reasoning trajectory, and then examine how each uses additional inference compute.

We introduce a controlled Countdown benchmark (Park et al., 2025) that holds out novel hierarchical combinations of familiar arithmetic patterns. Training Recursive and Linearized models on the same verified search trees under matched training budgets and shared inference token limits, we find that recursive execution improves accuracy while generating fewer tokens. We hypothesize that additional search helps discover unfamiliar compositions, motivating experiments on search



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.

scaling and the training widths needed to support it. We then improve search efficiency through call allocation across recursion levels and mixed-budget training. Finally, we use known compositional dependencies and current validation errors to adapt the training distribution toward patterns expected to support held-out compositions.

Our contributions are as follows:

- We introduce a controlled Countdown benchmark for evaluating generalization to unseen hierarchical compositions of familiar reasoning patterns.
- We show that recursive execution improves accuracy and generated token efficiency under matched training supervision, and study how adjusting and scaling inference budget affects this trade-off.
- We develop a composition-aware curriculum that improves transfer to patterns absent from the training examples.

## 2 Related work

**Reasoning and compositional generalization.** Generalization in reasoning-heavy tasks requires models to find the correct solution when a problem changes in its surface details, scale, or underlying structure (Mirzadeh et al., 2025; Lee et al., 2025; Zhao et al., 2024). Saparov et al. (2023) combine deduction rules in generated proof trees, but study transfer from in-context examples rather than a controlled train-test split. Sun et al. (2025) pair mathematical strategies from different problem families, without defining composition between their solution structures. In program synthesis, Shi et al. (2024) test unseen combinations and orders of operations, but note that a held-out program can yield a task solvable by a training-distribution program. Closest to our setting, Park et al. (2025) represent Countdown solutions as canonical expression trees and test unseen patterns at a fixed size. However, their holdout does not specifically test substitution between two retained training patterns.

**Adaptive curriculum and data mixing.** Once training and held-out patterns are separated, curriculum learning provides a way to shape the distribution over available training patterns to promote transfer to held-out structures. With target compositions withheld from training, a curriculum must decide which available patterns are most useful for learning them. Early adaptive methods select tasks according to the learner's progress, without modeling how one task contributes to another (Graves et al., 2017; Matiisen et al., 2019). Later data-mixing methods account for such interactions: Skill-It estimates prerequisite relations between skills through probing, DoGE weights source domains by their estimated effect on target performance by gradient alignment, and Aioli updates estimates of the loss-mixture relationship during training (Chen et al., 2023; Fan et al., 2024; Chen et al., 2025). These methods infer relationships from training behavior, which can scale poorly with number of patterns. In our setting, the decomposition of a held-out pattern into training patterns supplies additional information about which skills may transfer.

**Recursive inference.** Recursive inference provides a structured way to allocate additional inference compute to compositional problems: a model can decompose a problem into simpler patterns, solve them separately, and compose their results. Prior systems have used this structure to parallelize reasoning or manage long contexts (Schroeder et al., 2025; Zhang et al., 2025; Luo et al., 2025; Lian et al., 2025). Closest to our search supervision, APR trains models on search trajectories with `spawn` and `join` operations, emphasizing adaptive parallel search and latency on Countdown (Pan et al., 2025). Our execution uses nested calls, as in Yang et al. (2026), who study long-horizon reasoning and extrapolation to longer searches. We study a different question: whether recursive execution improves transfer to held-out compositions of familiar patterns, and how inference budgets and the training distribution affect that transfer.

## 3 Hierarchical Compositional Generalization

Compositional generalization is the systematic ability to understand or produce novel combinations of familiar constituents, a property classically associated with the compositionality and systematicity of cognition (Fodor & Pylyshyn, 1988; Lake & Baroni, 2018; Keysers et al., 2020). Its simplest form can be illustrated by independently varying attributes: after observing several colors and shapes, a

model should generalize to a held-out color-shape pair from their Cartesian product. Reasoning tasks often exhibit a richer, hierarchical form of composition. In theorem proving, the conclusion of one lemma can fill a typed premise of another; similarly, a program can pass the result of one subroutine into a particular argument of another (see fig. 1). Repeating this operation produces nested proof or call structures rather than flat combinations (de Moura & Ullrich, 2021; Ellis et al., 2023). We define the hierarchical composition the following way.

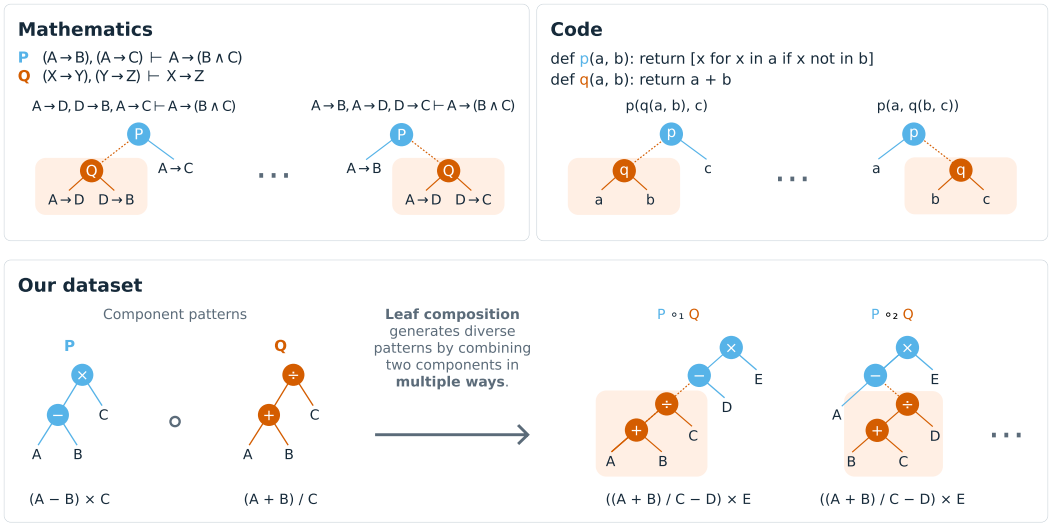



Figure 1: Hierarchical composition in mathematics, programming, and our dataset. The same outer component $P$ and inserted component $Q$ yield different structures when $Q$ is substituted at different input positions of $P$. Our dataset realizes this operation as leaf substitution in arithmetic expression trees. In particular, we substitute one of the leaves of arithmetic expression tree $P$ (the variable input) with the root of $Q$, creating a nested structure.

**Definition 1** (Hierarchical composition)**.** Let $A$ be a set, and let $\mathcal{P}$ be a set of patterns, where each $p \in \mathcal{P}$ denotes a function $p\colon A^{m_p} \to A$ of $m_p$ arguments. For two patterns $p\colon A^{m_p} \to A$, $q\colon A^{m_q} \to A$, and an argument position $i$, we define their composition as

$$(p \circ_i q)(x_1, \ldots, x_{m_p+m_q-1}) = p\big(x_1, \ldots, x_{i-1}, q(x_i, \ldots, x_{i+m_q-1}), x_{i+m_q}, \ldots, x_{m_p+m_q-1}\big).$$

Thus, $p \circ_i q\colon A^{m_p+m_q-1} \to A$ substitutes the input of $p$ at position $i$ with the output of $q$. For $\mathcal{S} \subseteq \mathcal{P}$, let $\mathcal{C}^2(\mathcal{S}) = \{p \circ_i q : p, q \in \mathcal{S},\ i \text{ valid}\}$ be its pairwise compositions.

To study hierarchical compositional generalization, we employ Countdown as a controlled domain in which reasoning structures can be enumerated and composed (Park et al., 2025). A problem consists of a list of input integers and a target. A valid solution is an arithmetic expression that uses every input exactly once and evaluates to the target. Following Park et al. (2025), we define a *pattern* by replacing the numerical leaves of a solution with placeholders and canonicalizing its expression tree, so different problems can instantiate the same structure; for example, $9/3 + 1$ and $12/4 + 2$ share the pattern $A/B + C$, see fig. 1. Patterns compose by replacing one input position with another pattern, mirroring how the output of a subroutine can fill an argument or a lemma can support a proof step. We restrict the benchmark to patterns with at most five leaves, select a set of training patterns $\mathcal{P}_{\text{train}}$, and define the pool of compositional patterns for evaluation as

$$\widehat{\mathcal{P}}_{\text{comp}} = \big(\mathcal{C}^2(\mathcal{P}_{\text{train}}) \setminus \mathcal{P}_{\text{train}}\big) \cap \mathcal{P}_5.$$

Each compositional pattern therefore combines two constituent patterns from training, while the complete five-leaf pattern is absent from training. Validation (Seen pattern) and Test (Seen pattern) contain new numerical instantiations of patterns in $\mathcal{P}_{\text{train}}$; Validation (Comp. pattern) and Test (Comp. pattern) instantiate the respective held-out subsets of $\widehat{\mathcal{P}}_{\text{comp}}$. Because a numerical problem may admit multiple solutions, we exclude a compositional instance whenever any training pattern can also solve it. Appendix section A.1 provides the complete generation and filtering procedure.

**Compositional ratio.** For a held-out pattern $p$, let $\mathcal{D}(p)$ contain all decompositions $p = l \circ_i r$ with $l, r \in \mathcal{P}_{\text{train}}$. For model $M$, we define

$$\text{CR}_M(p) = \frac{\text{pass@1}_M(p)}{\max_{(l,r,i)\in\mathcal{D}(p)} [\text{pass@1}_M(l) \times \text{pass@1}_M(r)]}$$

where $\text{pass@1}_M(p)$ is the accuracy of model $M$ on the pattern $p$. The ratio compares accuracy on a composition with factorized accuracy on its strongest pair of constituent patterns, and measures a compositional gap for each particular pattern.

## 4 Recursive Language Model

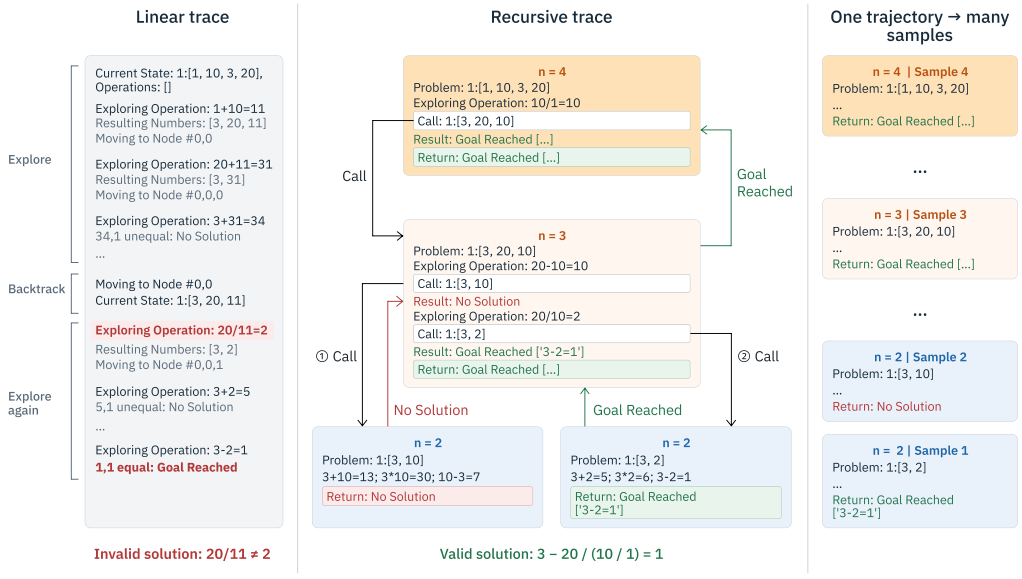



Figure 2: **(Left)**: A linearized trajectory interleaves exploration and backtracking in a single context, reaching an invalid solution after the arithmetic error $20/11 = 2$. **(Middle)**: Recursive execution separates subproblems into distinct contexts, linked by calls (black) and returns indicating exhausted search (red) or success (green). **(Right)**: Illustration of the training format: each context becomes a separate sample. Trajectories are abridged generated examples for the same problem.

Our implementation uses the subcall-based search of APR (Pan et al., 2025) with the nested, isolated contexts of Yang et al. (2026). The model searches over Countdown states and may emit a `Call` to solve a reduced state. A controller suspends the parent thread, runs the same model on that state, and inserts the child's `Return` as a `Result` before the parent continues. Children can schedule further calls recursively, yielding a depth-first execution tree. Because each parent receives results rather than its children's search histories, the overall trajectory may be longer than any individual context. The controller enforces execution limits but does not select arithmetic operations (fig. 2).

**Inference-time scaling.** We study how the number and placement of recursive calls affect compositional generalization. Each context with $n$ numbers has a local call budget $b_n$, with every child call using one unit. We write the budgets for five-, four-, and three-number problems as $b_5/b_4/b_3$, while two-number contexts enumerate terminal operations. For a five-number problem, the maximum total number of child calls is

$$C(b_5, b_4, b_3) = b_5 + b_5 b_4 + b_5 b_4 b_3.$$

Larger budgets permit wider search, while varying them across levels changes where that search occurs. If the model requests a call beyond its local budget, the controller returns `Search`

`Exhausted` to the parent, indicating a resource limit rather than an unsolvable subproblem. A global token limit across all calls lets us compare this form of scaling with extending a single linearized trace.

### 4.1 TRAINING RLM

We train with supervised fine-tuning on verified Countdown search trees. Following the SoS+ strategy of Pan et al. (2025), which extends Stream of Search (Gandhi et al., 2024), a symbolic generator explores candidate arithmetic operations, including failed branches, before reaching a verified solution. As illustrated in fig. 2, Recursive splits each tree at call boundaries: each context becomes a training example, and child results inserted into the parent are excluded from the loss. This permits training on trees whose full transcripts exceed one context window. The procedure leaves two choices: (i) how much search to generate, and (ii) how to sample its contexts, since a full tree contains many more low-level subproblems than high-level ones. Beyond Countdown, applying this recipe would require subtrajectories that can be identified and checked, for example with proof checkers or code tests, or a separate reward model.

**Baselines.** Linearized places the same search tree in one context, while No Search retains only a verified solution path. In matched comparisons, the formats use the same problems and search trees and receive the same supervised-token budget.

### 4.2 ADAPTIVE COMPOSITIONAL CURRICULUM

The benchmark's compositional structure provides information about which training patterns could help solve held-out tasks: each held-out pattern has known decompositions into available constituents. This suggests allocating more training to constituents whose improvement is predicted to benefit the validation compositions. Our adaptive curriculum uses the task's decomposition structure and current validation errors to allocate training toward patterns expected to help unseen compositions. We define the validation objective over:

$$\mathcal{P}_{\text{val}} = \mathcal{P}_{\text{train}} \sqcup \mathcal{P}_{\text{comp}}^{\text{val}},$$

where the first set contains new numerical instantiations of training patterns and the second contains patterns absent from training. For a validation set $V_p$, we define the error of pattern $p$ at curriculum round $t$ as

$$\ell_{t,p} = 1 - \frac{1}{|V_p|} \sum_{x \in V_p} \mathbf{1}\{M_t \text{ solves } x\}.$$

Let $C$ denote total training compute and $C_p$ the compute allocated directly to pattern $p$. We describe its local learning dynamics by

$$\frac{\Delta \ell_p}{\Delta C_p} = f_p(\ell_p, C_p).$$

where $\Delta \ell_p = \ell_{t+1,p} - \ell_{t,p}$, i.e. the change in the error at the next round. Following Skill-It (Chen et al., 2023), we assume that this rate is proportional to the current error. At curriculum round $t$,

$$f_{t,p} := f_p(\ell_{t,p}, C_{t,p}) = -\kappa \ell_{t,p}, \qquad \kappa > 0.$$

An allocation $\mathbf{u} \in \mathcal{U}$ distributes the next $\Delta C$ training tokens across the training patterns, where $\mathcal{U}$ is the simplex over $\mathcal{P}_{\text{train}}$ and $u_p = 0$ for held-out patterns. Since $u_p = \Delta C_p / \Delta C$, the direct error dynamics are

$$\frac{\Delta \ell_p}{\Delta C} = f_{t,p} u_p.$$

To capture dependencies between patterns, let $A_t[p, q]$ describe how a change in the error of pattern $q$ contributes to the error change of pattern $p$. We assume the coupled dynamics

$$\frac{\Delta \boldsymbol{\ell}}{\Delta C} = \text{diag}(\mathbf{f}_t)\mathbf{u} + A_t \frac{\Delta \boldsymbol{\ell}}{\Delta C}$$

Using a first-order approximation over the next $\Delta C$ training tokens, we choose the allocation that minimizes the modeled change in mean validation error $\frac{1}{|\mathcal{P}_{\mathrm{val}}|}\sum_{p\in\mathcal{P}_{\mathrm{val}}}\Delta\ell_p$. Then

$$J_t(\mathbf{u}) = \frac{1}{|\mathcal{P}_{\mathrm{val}}|}\sum_{p\in\mathcal{P}_{\mathrm{val}}}\left[\Delta C\,(R_t\,\mathrm{diag}(\mathbf{f}_t)\mathbf{u})_p\right], \qquad R_t = (I - A_t)^{-1}$$

For each training pattern $a$, define its predicted improvement score

$$s_t(a) = -\frac{\Delta C\, f_{t,a}}{|\mathcal{P}_{\mathrm{val}}|}\sum_{p\in\mathcal{P}_{\mathrm{val}}} R_t[p,a] = \frac{\kappa\Delta C\,\ell_{t,a}}{|\mathcal{P}_{\mathrm{val}}|}\sum_{p\in\mathcal{P}_{\mathrm{val}}} R_t[p,a].$$

**Theorem 1** (Greedy and smooth allocations)**.** Under the locally frozen dynamics above, the greedy allocation minimizing $J_t(\mathbf{u})$ selects a pattern with the largest predicted improvement:

$$a_t^{\star,\mathrm{greedy}} \in \arg\max_{a\in\mathcal{P}_{\mathrm{train}}} s_t(a), \qquad \mathbf{u}_t^{\star,\mathrm{greedy}} = \mathbf{e}_{a_t^{\star,\mathrm{greedy}}},$$

where $\mathbf{e}_a$ assigns the complete next-round budget to pattern $a$. If several patterns attain the maximum, any allocation supported on those patterns is optimal.

For a smooth allocation, let $\mathbf{u}_{t-1}$ be a strictly positive previous allocation and define

$$\mathbf{u}_t^{\star,\mathrm{smooth}} = \arg\min_{\mathbf{u}\in\mathcal{U}}\left\{J_t(\mathbf{u}) + \frac{1}{\eta}D_{\mathrm{KL}}(\mathbf{u}\|\mathbf{u}_{t-1})\right\}.$$

Its unique solution is

$$u_t^{\star,\mathrm{smooth}}(a) = \frac{u_{t-1}(a)\exp(\eta s_t(a))}{\sum_{b\in\mathcal{P}_{\mathrm{train}}} u_{t-1}(b)\exp(\eta s_t(b))}.$$

**Propagation through $R_t$.** The inverse $R_t = (I - A_t)^{-1} = I + A_t + A_t^2 + \cdots + A_t^H$ accounts for direct and indirect relationships between patterns. The identity term represents direct learning, while $A_t$ captures effects between directly related patterns. If $p_1$ contributes to $p_2$ and $p_2$ contributes to $p_3$, then $A_t^2$ captures the indirect effect of $p_1$ on $p_3$. Higher powers represent longer compositional paths. Appendix section A.3 provides the derivations and proofs.

**Composition-induced dependencies.** We derive $A_t$ from the known relationships between patterns. For a pattern $p$, let $\mathcal{D}(p) = \{(l, r, i) : p = l \circ_i r\}$ contain its admissible decompositions, and let $\pi_t(d \mid p)$ be nonnegative weights that sum to one over $d \in \mathcal{D}(p)$. We use uniform weights when no decomposition preference is available.

Conditioned on a decomposition $d = (l, r, i)$, solving $p$ requires solving both constituents and composing their results. Assuming independent constituent successes, its success probability is approximately $(1 - \ell_l)(1 - \ell_r)$. Averaging over decompositions yields the compositional error predictor

$$G_{t,p}(\boldsymbol{\ell}) = 1 - \sum_{(l,r,i)\in\mathcal{D}(p)} \pi_t(l, r, i \mid p)(1 - \ell_l)(1 - \ell_r).$$

We define the local response matrix as its Jacobian:

$$A_t[p, q] = \left.\frac{\partial G_{t,p}}{\partial \ell_q}\right|_{\boldsymbol{\ell}=\boldsymbol{\ell}_t}$$

**Theorem 2** (Composition-induced response)**.** The response matrix induced by $G_{t,p}$ is

$$A_t[p, q] = \sum_{(l,r,i)\in\mathcal{D}(p)} \pi_t(l, r, i \mid p)\left[\mathbf{1}\{q = l\}(1 - \ell_{t,r}) + \mathbf{1}\{q = r\}(1 - \ell_{t,l})\right].$$

For this construction, $I - A_t$ is invertible. If $H$ is the longest constituent-to-parent path, then

$$(I - A_t)^{-1} = I + A_t + A_t^2 + \cdots + A_t^H.$$

Every dependency connects a smaller constituent to a larger parent, making the compositional graph acyclic. Appendix A.3 provides the full derivation and invertibility proof.

At each curriculum round, we evaluate the per-pattern errors, construct $A_t$, compute the scores $s_t(a)$, and sample the next training block using the greedy or smooth allocation from Theorem 1. Compositional validation and test patterns are disjoint: the curriculum uses feedback on validation compositions, while final evaluation measures transfer to different, unseen compositions.

# 5 RESULTS

We organize the experiments around three questions:

- **RQ1:** Does recursive inference improve compositional generalization over a matched linearized search?
- **RQ2:** How does compositional generalization improve with inference-time compute, and how can we make it more efficient?
- **RQ3:** Can the knowledge of compositional structure be utilized to improve generalization to held-out compositions?

We fine-tune SmolLM2-135M, Gemma3-270M, and SmolLM2-1.7B on approximately 1B supervised tokens each, covering two model families and three sizes. Checkpoints are selected by mean accuracy across the five validation groups. Within each model, Recursive and Linearized use the same underlying training problems and verified search trees, rendered in their respective formats, with matched supervised-token budgets. We report pass@1 under greedy decoding on Test (Seen pattern) and Test (Comp. pattern), together with mean generated tokens and the compositional ratio.

## 5.1 RQ1: DOES RECURSIVE EXECUTION IMPROVE COMPOSITIONAL GENERALIZATION?

We compare Linearized and Recursive on the same problems and $3/3/3$ search trees using SmolLM2-135M, Gemma3-270M, and SmolLM2-1.7B. Within each model, the formats use the same data construction and matched training budget.

Table 1: Test pass@1, compositional ratio, and mean generated tokens at the 8,192-token limit. Linearized and Recursive use the same problems and $3/3/3$ search trees within each model.

| Model | Format | Test Seen | Test Comp. | Comp. ratio | Tokens Seen / Comp. |
|---|---|---|---|---|---|
| SmolLM2-135M | Linearized | .068 | .010 | .014 | 7,117 / 7,580 |
| | Recursive | **.160** | **.042** | **.083** | **4,562 / 4,927** |
| Gemma3-270M | Linearized | .173 | .044 | .060 | 7,135 / 7,705 |
| | Recursive | **.180** | **.086** | **.154** | **5,977 / 6,428** |
| SmolLM2-1.7B | Linearized | .121 | .015 | .019 | 7,117 / 7,763 |
| | Recursive | **.267** | **.107** | **.164** | **4,393 / 4,859** |

**Recursive obtains higher compositional accuracy with fewer generated tokens** for all three models (table 1). Its compositional pass@1 is approximately twice that of Linearized for Gemma3-270M, four times higher for SmolLM2-135M, and seven times higher for SmolLM2-1.7B.

## 5.2 RQ2: HOW DOES COMPOSITIONAL GENERALIZATION IMPROVE WITH INFERENCE BUDGET SCALING?

Our hypothesis is that additional search can help a model discover unfamiliar combinations of known operations. We first test whether increasing recursive search improves held-out compositional accuracy, and whether training on wider searches enables the model to use larger inference budgets. We then ask how to obtain these gains with fewer generated tokens by changing the allocation of calls across recursion levels and the mixture of search widths used during training.

**Scaling width rather than trace length.** Linearized scales inference compute by generating a longer autoregressive trajectory, whereas Recursive scales it by allowing more calls across separate contexts. Figure 3 compares the two mechanisms. Extending Linearized from 8,192 to 16,384 tokens raises both seen and compositional pass@1 about 10% ($.068 \rightarrow .075$ on seen patterns and $.010 \rightarrow .011$ on compositional held-out ones). In contrast, Recursive already reaches .160 and .042 with the $3/3/3$ policy while generating approximately 5,000 tokens. Training Recursive on larger search trees, with the same supervised-token budget, allows it to learn from searches whose aggregate trajectories exceed one context. Its full search reaches .235 seen-pattern and .119 compositional pass@1.

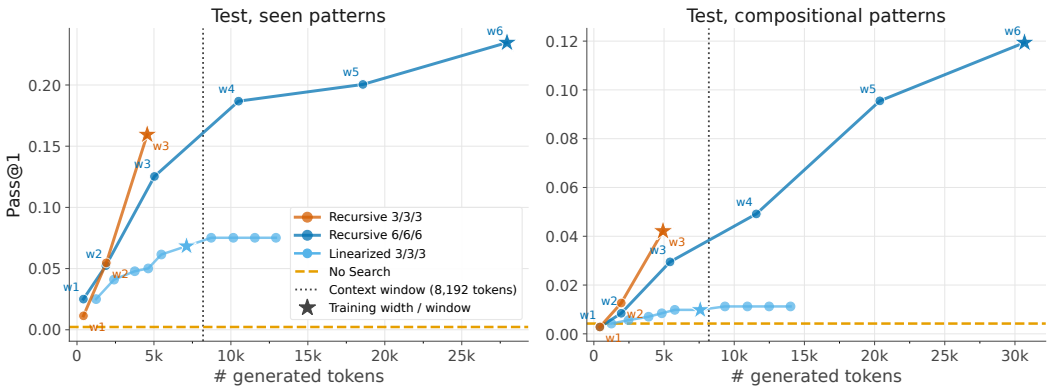



Figure 3: Test pass@1 against mean generated tokens. Linearized scales by extending one trajectory; Recursive scales by widening the controller. Stars mark the search width or token limit used during training.

**Allocating calls across depth levels.** Wider search is costly, and the same call allowance can be distributed differently across recursion levels. We compare policies with approximately the same maximum number of calls but different allocations across recursion levels. Table 2 reports full-search results.

Table 2: Recursive inference under approximately matched call allowances. Each model is trained and evaluated using the indicated allocation.

| Allocation | Test Seen | Test Comp. | Comp. ratio | Tokens Seen / Comp. |
|---|---|---|---|---|
| $10/5/4$, root-heavy | .248 | .093 | **.219** | **23,248 / 28,388** |
| $6/6/6$, uniform | .235 | **.119** | .186 | 27,940 / 30,680 |
| $6/7/5$, middle-heavy | **.273** | .101 | .183 | 27,021 / 30,413 |
| $4/7/8$, leaf-heavy | .189 | .072 | .127 | 29,355 / 32,560 |

Root-heavy allocation uses the fewest tokens and attains the highest compositional ratio, while leaf-heavy allocation is worse in both accuracy and token use. Uniform allocation obtains the highest raw compositional accuracy, and middle-heavy allocation obtains the highest seen-pattern accuracy. Thus, allocating more calls near the root is more efficient than concentrating them near the leaves, though no allocation dominates across all metrics.

**Mixing widths during training.** Finally, a complementary way to improve efficiency is to train one model to operate across several search budgets. We compare a model trained only on $10/5/4$ trees with one trained on an equal-token mixture of $6/3/2$, $8/4/3$, and $10/5/4$ trees. The mixed model improves the accuracy–compute Pareto frontier across the evaluated controllers (fig. 4). At $8/4/3$, it reaches .269 seen-pattern and .089 compositional pass@1. This approximately matches the single-width model at $10/5/4$ while using half as many generated tokens (14k vs. 28k).

The mixed model also exhibits modest raw gains beyond its largest training call budget: compositional pass@1 increases .096 $\rightarrow$ .107 when scaling inference compute from $10/5/4$ to $15/8/6$. Mixed-width training clearly improves compute efficiency, while systematic extrapolation beyond the largest training width remains limited.

### 5.3 RQ3: Does a composition-aware curriculum improve generalization to held-out compositions?

**The curriculum nearly doubles held-out compositional pass@1**, from .042 to .083, while increasing seen-pattern accuracy from .160 to .205 (table 3), with the $3/3/3$ inference search budgets unchanged. Relative to this fixed-policy reference, the curriculum raises compositional accuracy

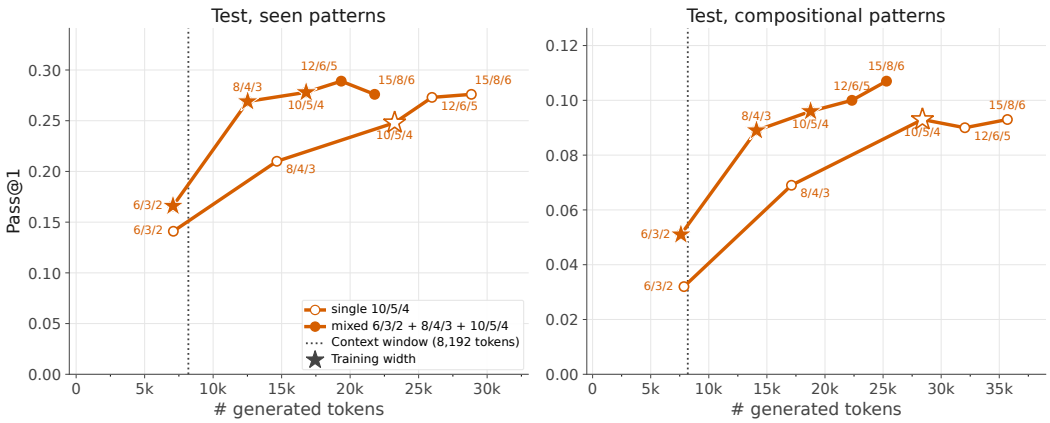



Figure 4: Test pass@1 against mean generated tokens for a model trained only at $10/5/4$ and a model trained on a mixture of $6/3/2$, $8/4/3$, and $10/5/4$. Stars mark training widths.

Table 3: Test pass@1 for SmolLM2-135M trained on (3/3/3) search trees with and without curriculum. The fixed-policy reference reports the accuracy of the search policy (SoS+) used to generate the data within budget.

| Baseline | Test (Seen pattern) | Test (Comp. pattern) |
|---|---|---|
| Linearized | .068 | .010 |
| Recursive | .160 | .042 |
| Recursive + Curriculum | **.205** | **.083** |
| Fixed-policy reference | .262 | .131 |

from approximately 32% to 63% of the reference value, and seen-pattern accuracy from 61% to 78%. An important implementation choice is subtrajectory sampling: allocating training to a root pattern does not directly determine the distribution of subproblems seen by the model. Larger search trees contain many lower-level contexts, so retaining every context can concentrate supervision on small subproblems even when the curriculum selects larger patterns. We therefore retain roots, solution paths, and successful return chains while sampling the remaining contexts to control the token distribution across subproblem sizes. Appendix section A.2 details this procedure and compares it with full-tree training, alternative token allocations and curriculum step sizes.

## 6 Conclusion

We studied hierarchical compositional generalization on a controlled Countdown benchmark. Under matched search supervision and training budgets, recursive execution improved accuracy while generating fewer tokens than linearized execution. Training and inference with wider recursive search further improved generalization, while call allocation and mixed-budget training improved token efficiency. A composition-aware curriculum doubled held-out compositional accuracy, showing how known structural dependencies can guide training.

Countdown isolates tree composition and excludes held-out problems solvable through seen patterns, but omits much of the complexity of real-world reasoning. Our results are limited to relatively small models, and the curriculum requires explicit compositional structure. Whether these gains extend to larger models and richer domains where such structure must be inferred remains open.

## 7 AI USE STATEMENT

We used generative AI tools to assist with drafting and revising the manuscript, improving its organization and clarity, as well as creating figures. The AI tools have also been used for finding related literature, code generation and experiments ideation. The exact method derivations and hierarchical composition definition were polished with generative AI.

## 8 REPRODUCIBILITY STATEMENT

Section 3 defines the compositional benchmark and evaluation metric, and Section A.1 describes pattern construction, numerical instantiation, and dataset filtering. Section 4 describes recursive execution, search budgets, training formats, and the adaptive curriculum. Section 5 reports the experimental setup and evaluation protocol. The details for curriculum are reported in Appendix Section A.2.

## REFERENCES

Mayee Chen, Nicholas Roberts, Kush Bhatia, Jue Wang, Ce Zhang, Frederic Sala, and Christopher Ré. Skill-it! a data-driven skills framework for understanding and training language models. *Advances in Neural Information Processing Systems*, 36:36000–36040, 2023.

Mayee Chen, Michael Hu, Nicholas Lourie, Kyunghyun Cho, and Christopher Ré. Aioli: A unified optimization framework for language model data mixing. In *International Conference on Learning Representations*, volume 2025, pp. 46089–46132, 2025.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pp. 625–635. Springer, 2021. doi: 10.1007/978-3-030-79876-5_37.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-3081.

Kevin Ellis, Lionel Wong, Maxwell Nye, Mathias Sablé-Meyer, Luc Cary, Lore Anaya Pozo, Luke Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum. DreamCoder: Growing generalizable, interpretable knowledge with wake–sleep bayesian program learning. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 381(2251): 20220050, 2023. doi: 10.1098/rsta.2022.0050.

Simin Fan, Matteo Pagliardini, and Martin Jaggi. DOGE: Domain reweighting with generalization estimation. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=7rfZ6bMZq4.

Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1–2):3–71, 1988. doi: 10.1016/0010-0277(88)90031-5.

Kanishk Gandhi, Denise H J Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah Goodman. Stream of search (sos): Learning to search in language. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=2cop2jmQVL.

Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *international conference on machine learning*, pp. 1311–1320. Pmlr, 2017.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SygcCnNKwr.

Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 2873–2882. PMLR, 2018. URL https://proceedings.mlr.press/v80/lake18a.html.

Nayoung Lee, Ziyang Cai, Avi Schwarzschild, Kangwook Lee, and Dimitris Papailiopoulos. Self-improving transformers overcome easy-to-hard and length generalization challenges. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=ZtX0MBT6mf.

Long Lian, Sida Wang, Felix Juefei-Xu, Tsu-Jui Fu, Xiuyu Li, Adam Yala, Trevor Darrell, Alane Suhr, Yuandong Tian, and Xi Victoria Lin. ThreadWeaver: Adaptive threading for efficient parallel reasoning in language models, 2025. URL https://arxiv.org/abs/2512.07843.

Hongyin Luo, Nathaniel Morgan, Tina Li, Derek Zhao, Ai Vy Ngo, Philip Schroeder, Lijie Yang, Assaf Ben-Kish, Jack O'Brien, and James Glass. Beyond context limits: Subconscious threads for long-horizon reasoning. *arXiv preprint arXiv:2507.16784*, 2025.

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher–student curriculum learning. *IEEE transactions on neural networks and learning systems*, 31(9):3732–3740, 2019.

Seyed Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-Symbolic: Understanding the limitations of mathematical reasoning in large language models. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=AjXkRZIvjB.

Jiayi Pan, Xiuyu Li, Long Lian, Charlie Victor Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. Learning adaptive parallel reasoning with language models. In *Second Conference on Language Modeling*, 2025.

Simon Park, Simran Kaur, and Sanjeev Arora. How does RL post-training induce skill composition? a case study on Countdown, 2025. URL https://arxiv.org/abs/2512.01775.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-emnlp.378.

Abulhair Saparov, Richard Yuanzhe Pang, Vishakh Padmakumar, Nitish Joshi, Mehran Kazemi, Najoung Kim, and He He. Testing the general deductive reasoning capacity of large language models using ood examples. *Advances in Neural Information Processing Systems*, 36:3083–3105, 2023.

Philip Schroeder, Nathaniel W. Morgan, Hongyin Luo, and James R. Glass. THREAD: Thinking deeper with recursive spawning. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8418–8442. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.427.

Kensen Shi, Joey Hong, Yinlin Deng, Pengcheng Yin, Manzil Zaheer, and Charles Sutton. Exedec: Execution decomposition for compositional generalization in neural program synthesis. In *International Conference on Learning Representations*, volume 2024, pp. 45106–45135, 2024.

Yiyou Sun, Shawn Hu, Georgia Zhou, Ken Zheng, Hanna Hajishirzi, Nouha Dziri, and Dawn Song. OMEGA: Can LLMs reason outside the box in math? evaluating exploratory, compositional, and transformative generalization. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, volume 38, 2025.

Xisheng Xiao and Hanlin Zhao. From A and B to A+B: Can large language models solve compositional math problems? In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13057–13078. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.660.

Chenxiao Yang, Nathan Srebro, and Zhiyuan Li. Recursive models for long-horizon reasoning. *arXiv preprint arXiv:2603.02112*, 2026.

Alex L Zhang, Tim Kraska, and Omar Khattab. Recursive language models. *arXiv preprint arXiv:2512.24601*, 2025.

Haoyu Zhao, Simran Kaur, Dingli Yu, Anirudh Goyal, and Sanjeev Arora. Can models learn skill composition from examples? In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-3251.

# A Appendix

## A.1 Dataset Generation

In this section, we describe how we enumerate Countdown patterns, generate their numerical instantiations, and construct training, validation, and test sets that isolate hierarchical compositional generalization. We also describe how training problems are sampled to ensure that the execution formats receive the same underlying problems and search trees.

Table 4: Countdown training and evaluation splits. Training examples are sampled to a fixed token budget; the $n = 5$ pool contains 235,126 eligible problems. Test sets contain 48 seen and 82 compositional patterns.

| Partition | $n$ | Pattern set | # Problems |
|---|---|---|---|
| Training pool | 2–5 | $\mathcal{P}_{\text{train}}$ | sampled with replacement |
| Validation (Seen) | 2 | $\mathcal{P}_{\text{train}}$ | 73 |
| Validation (Seen) | 3 | $\mathcal{P}_{\text{train}}$ | 348 |
| Validation (Seen) | 4 | $\mathcal{P}_{\text{train}}$ | 911 |
| Validation (Seen) | 5 | $\mathcal{P}_{\text{train}}$ | 384 |
| Validation (Compositional) | 5 | $\mathcal{P}_{\text{comp}}$ | 732 |
| Test (Seen) | 5 | $\mathcal{P}_{\text{train}}$ | 439 |
| Test (Compositional) | 5 | $\mathcal{P}_{\text{comp}}$ | 712 |

**Pattern space.** To construct training and evaluation splits that differ in reasoning structure rather than numerical values, we first enumerate the space of Countdown solution patterns. Each pattern is an expression tree whose leaves are ordered input placeholders and whose internal nodes are addition, subtraction, multiplication, or division. We consider patterns with two to five leaves and canonicalize them under arithmetic symmetries following Park et al. (2025). The four two-input patterns $\mathcal{P}_{\text{atom}} = \{A + B, A - B, A \times B, A/B\}$ generate the larger pattern space through repeated leaf substitution. After composing two patterns, we canonicalize the resulting tree and retain it only if the inserted pattern still appears as a distinct subtree, not present in the training set.

Unlike typed programs or proofs, all Countdown expressions return the same numerical type, so any pattern can structurally fill any input position. In Lean, by contrast, a proof term can fill a goal only when its type matches the expected proposition, dependent types may impose additional context-sensitive constraints, and hypotheses may be reused or ignored; Countdown therefore isolates tree composition while omitting much of the typing and resource structure of realistic reasoning domains (de Moura & Ullrich, 2021).

**Numerical instantiation.** Problems are generated pattern-first. After selecting a pattern, we sample values for its leaves from ([1,50]), evaluate the expression, and retain the instance when every intermediate operation produces a well-defined integer and the final target lies in ([1,9999]). This integer-only restriction differs from the construction of Park et al. (2025), which permits rational intermediate values, and matches the arithmetic semantics of our search procedure. A numerical problem may admit several valid solution patterns even though it was generated from one particular pattern.

$$\left(\mathcal{C}^2(\mathcal{P}_{\text{train}}) \setminus \mathcal{P}_{\text{train}}\right) \cap \mathcal{P}_5.$$

Every pattern in $\mathcal{P}_{\text{comp}}$ combines two constituent patterns from training, while the complete five-leaf pattern is absent from training. Candidate compositional patterns are ordered with a fixed seed and stratified by tree shape, root operation, and the sizes of the two root subtrees before being divided into disjoint validation and test subsets. *Validation (Seen pattern)* and *Test (Seen pattern)* contain new numerical instantiations of patterns in $\mathcal{P}_{\text{train}}$. *Validation (Comp. pattern)* and *Test (Comp. pattern)* instantiate the corresponding held-out subsets of $\mathcal{P}_{\text{comp}}$. Numerical instances are disjoint across training, validation, and test.

Because the same inputs and target may admit several solutions, we additionally test every candidate compositional instance against every training pattern over all assignments of its input numbers. We reject the instance if any pattern in $\mathcal{P}_{\text{train}}$ reaches the target. Consequently, a compositional evaluation problem cannot be solved through a seen pattern, although it may admit multiple held-out solution patterns. Dataset statistics are reported in table 4.

**Training-pool sampling.** Training covers problem sizes (n=2,. . . ,5). Each size receives an equal share of the supervised-token budget, and patterns are sampled uniformly with replacement within each size. We exclude problems whose linearized (3/3/3) search transcript exceeds the model context window from the eligible pool of both execution formats. Each sampled problem has one generated search tree, which is rendered either as a single Linearized transcript or as separate Recursive contexts. All matched training baselines therefore use the same eligible problems, tree-generation procedure, and supervised-token budget.

### A.2 Curriculum ablations

All runs use Recursive SmolLM2-135M, $3/3/3$ search trees, and the smooth multiplicative allocation update, with reference step size $\eta = .006$. We compare settings after 15 rounds of approximately 32M supervised tokens each (about 480M tokens in total), using the learning-rate schedule of the full run. The first round samples root patterns uniformly. There is one run per setting. We retain the reference unless an alternative increases mean validation accuracy and the paired 95% intervals for its differences on both five-number groups have nonnegative lower bounds.

After each round, we evaluate per-pattern errors. For training patterns without validation examples, we add ten problems per pattern from the training pool, giving 5,208 evaluation problems. These extra probes inform the curriculum but are excluded from the validation score used for selection. Tables report problem-averaged accuracy within each group; Mean is the unweighted average over the five standard groups. Confidence intervals use a paired bootstrap of pattern-averaged accuracies and do not measure variation across training seeds.

**Subproblem sampling.** The curriculum selects root problems, while subproblem sampling controls which model contexts within each search tree are used for training. Full trees allocate 64% of supervised tokens to two-number problems and only 3% to five-number problems. We retain roots, the solution path, and successful return chains, and sample the remaining contexts to target cumulative token shares of $50/25/15/10$ percent at $n = 2/3/4/5$. Retention rates are adjusted before each round within $[.05, 1]$ using the tokens already consumed.

Table 5: Subproblem sampling: validation pass@1 after about 480M supervised tokens, with $\eta = .006$. Sampling targets a $50/25/15/10$ cumulative token split across $n = 2/3/4/5$.

| Setting | Mean | $n = 5$ Seen | $n = 5$ Comp. |
|---|---|---|---|
| Full trees | .392 | .117 | .010 |
| Sampled subproblems | **.455** | **.180** | **.045** |

Sampling improves mean validation accuracy from .392 to .455 in this comparison (table 5). Both five-number groups improve, with paired 95% pattern-bootstrap intervals excluding zero.[1]

We also test $40/30/18/12$ and $35/32/20/13$ token splits. Across the three sampled settings, mean validation accuracy ranges from .454 to .457. Neither alternative meets the selection criterion: both

[1]The full-tree control suffered a loss spike, which may explain part of this gap.

lower five-number seen accuracy, and all paired intervals for the five-number groups include zero. We retain $50/25/15/10$.

**Step-size sensitivity.** We halve and double $\eta = .006$, keeping the $50/25/15/10$ sampling target. Each run starts from the base model.

Table 6: Step-size sensitivity: validation pass@1 after about 480M supervised tokens, with the same subproblem-sampling target.

| $\eta$ | Mean | $n = 5$ Seen | $n = 5$ Comp. |
|---|---|---|---|
| .003 | .447 | .169 | **.048** |
| .006 | **.455** | **.180** | .045 |
| .012 | .446 | .169 | .037 |

Halving or doubling the step size yields similar mean validation accuracy in these runs (table 6). For both five-number groups, paired 95% pattern-bootstrap intervals for differences from the reference include zero. We retain $\eta = .006$, which has the highest mean accuracy.

**Final training and allocation.** We continue the selected recipe to approximately 1B supervised tokens. As for the other methods, we report the checkpoint with the best validation performance, measured here by the mean over the five standard groups (table 3).

During training, the share of three-number root problems rises from 4% to 36%, while the five-number share falls from 74% to 43% (fig. 5). These root-problem shares describe the adaptive allocation; they differ from the controlled token shares across sampled subproblem sizes.

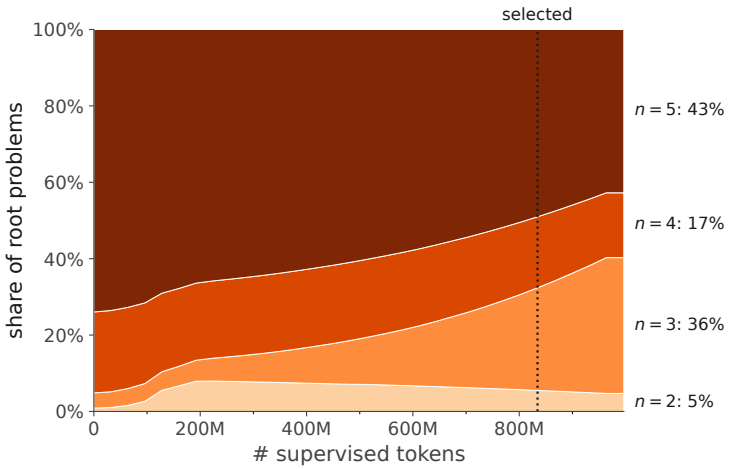



Figure 5: Root-problem shares by size during curriculum training, against supervised tokens. These are root-problem shares, not subproblem-token shares. The dotted line marks the validation-selected checkpoint; labels give final shares.

### A.3 Curriculum derivations and proofs

Let $\mathcal{P}$ contain the training and validation patterns together with any additional constituents required by their decompositions. All vectors and matrices below are indexed by $\mathcal{P}$, and we extend $u_p = 0$ to patterns outside $\mathcal{P}_{\text{train}}$. We model the change in validation error over one curriculum round, using $\Delta\ell_p = \ell_{t+1,p} - \ell_{t,p}$. The equations below describe a local approximation to learning over a finite block of training tokens. We hold the learning rates $\mathbf{f}_t$, response matrix $A_t$, and allocation $\mathbf{u}$ fixed within that round.

### A.3.1 FINITE-STEP ALLOCATION MODEL

Let $\Delta C$ be the supervised-token budget for the next round and $\Delta C_p$ the budget allocated directly to pattern $p$. Then

$$\Delta C_p = u_p \Delta C.$$

With direct learning alone, we model the error change as

$$\Delta \ell_p = f_{t,p} \Delta C_p = \Delta C \, f_{t,p} u_p, \qquad f_{t,p} = -\kappa \ell_{t,p}.$$

The coupled model adds the effects of error changes in related patterns:

$$\Delta \boldsymbol{\ell} = \Delta C \ \mathrm{diag}(\mathbf{f}_t)\mathbf{u} + A_t \Delta \boldsymbol{\ell}.$$

Rearranging yields

$$(I - A_t)\Delta \boldsymbol{\ell} = \Delta C \ \mathrm{diag}(\mathbf{f}_t)\mathbf{u}.$$

The invertibility of $I - A_t$ for our compositional construction is proved below. Defining $R_t = (I - A_t)^{-1}$, we obtain

$$\Delta \boldsymbol{\ell} = \Delta C \, R_t \, \mathrm{diag}(\mathbf{f}_t)\mathbf{u}.$$

Averaging the modeled error change over validation patterns gives the objective used in section 4.2:

$$J_t(\mathbf{u}) = \frac{1}{|\mathcal{P}_{\mathrm{val}}|} \sum_{p \in \mathcal{P}_{\mathrm{val}}} \Delta \ell_p = \frac{\Delta C}{|\mathcal{P}_{\mathrm{val}}|} \sum_{p \in \mathcal{P}_{\mathrm{val}}} \sum_{a \in \mathcal{P}_{\mathrm{train}}} R_t[p,a] f_{t,a} u_a.$$

Since the current validation error is fixed, minimizing its modeled change also minimizes the modeled error after the next round. Rearranging the sums,

$$J_t(\mathbf{u}) = - \sum_{a \in \mathcal{P}_{\mathrm{train}}} u_a s_t(a),$$

where

$$s_t(a) = -\frac{\Delta C \, f_{t,a}}{|\mathcal{P}_{\mathrm{val}}|} \sum_{p \in \mathcal{P}_{\mathrm{val}}} R_t[p,a].$$

Under $f_{t,a} = -\kappa \ell_{t,a}$, this becomes

$$s_t(a) = \frac{\kappa \Delta C \, \ell_{t,a}}{|\mathcal{P}_{\mathrm{val}}|} \sum_{p \in \mathcal{P}_{\mathrm{val}}} R_t[p,a].$$

### A.3.2 PROOF OF THE ALLOCATION THEOREM

*Proof of theorem 1.* Since $J_t(\mathbf{u}) = -\sum_a u_a s_t(a)$, minimizing $J_t$ is equivalent to

$$\max_{\mathbf{u} \in \mathcal{U}} \sum_{a \in \mathcal{P}_{\mathrm{train}}} u_a s_t(a).$$

Because $\mathcal{U}$ is the probability simplex, the maximum is attained by assigning all mass to a pattern with maximal score. If several patterns share the maximum, every allocation supported on those patterns has the same objective value.

For the smooth allocation with $\eta > 0$, form the Lagrangian

$$\mathcal{L}(\mathbf{u}, \lambda) = -\sum_a u_a s_t(a) + \frac{1}{\eta} \sum_a u_a \log \frac{u_a}{u_{t-1}(a)} + \lambda \left( \sum_a u_a - 1 \right).$$

For each $a$, the first-order condition is

$$-s_t(a) + \frac{1}{\eta} \left( \log \frac{u_a}{u_{t-1}(a)} + 1 \right) + \lambda = 0.$$

Consequently,

$$u_a = u_{t-1}(a) \exp(\eta s_t(a) - 1 - \eta\lambda).$$

Normalizing the allocation gives

$$u_t^{\star,\mathrm{smooth}}(a) = \frac{u_{t-1}(a) \exp(\eta s_t(a))}{\sum_{b \in \mathcal{P}_{\mathrm{train}}} u_{t-1}(b) \exp(\eta s_t(b))}.$$

Since the KL term is strictly convex when $\mathbf{u}_{t-1}$ is strictly positive, this solution is unique. □

### A.3.3 PROOF OF THE COMPOSITION-INDUCED DEPENDENCY MATRIX

*Proof of theorem 2.* For a decomposition $d = (l, r, i)$, the composition model gives error

$$1 - (1 - \ell_l)(1 - \ell_r) = \ell_l + \ell_r - \ell_l \ell_r.$$

If the constituent errors change by $\Delta\ell_l$ and $\Delta\ell_r$, the change in this expression is exactly

$$(1 - \ell_{t,r})\Delta\ell_l + (1 - \ell_{t,l})\Delta\ell_r - \Delta\ell_l \Delta\ell_r.$$

We retain the terms linear in the error changes. Thus, a change in one constituent is weighted by the other constituent's current success probability. Holding $\pi_t$ fixed and averaging over decompositions gives the coefficient of each $\Delta\ell_q$:

$$A_t[p, q] = \sum_{(l,r,i)\in\mathcal{D}(p)} \pi_t(l, r, i \mid p) \left[\mathbf{1}\{q = l\}(1 - \ell_{t,r}) + \mathbf{1}\{q = r\}(1 - \ell_{t,l})\right].$$

If the same pattern occupies both constituent positions, both indicator terms contribute. For a pattern without an admissible decomposition, the corresponding row of $A_t$ is zero.

It remains to prove that $I - A_t$ is invertible. Direct an edge $q \to p$ whenever $q$ is a constituent of $p$ and $A_t[p, q] \neq 0$. Every proper constituent has fewer leaves than its parent. Ordering patterns by their number of leaves therefore makes $A_t$ strictly triangular, and the resulting dependency graph is acyclic.

More explicitly, matrix multiplication gives

$$(A_t^k)[p, q] = \sum_{q=p_0\to p_1\to\cdots\to p_k=p} \prod_{j=1}^{k} A_t[p_j, p_{j-1}],$$

where the sum ranges over all directed paths of length $k$ from $q$ to $p$. If the longest constituent-to-parent path has $H$ edges, no path of length $H + 1$ exists. Hence

$$A_t^{H+1} = 0.$$

The finite geometric identity then gives

$$(I - A_t)\left(I + A_t + \cdots + A_t^H\right) = I - A_t^{H+1} = I.$$

Therefore $I - A_t$ is invertible and

$$(I - A_t)^{-1} = I + A_t + \cdots + A_t^H.$$

□

### A.4 ALTERNATIVE FINITE-STEP LEARNING MODELS

The proportional learning model assumes that a finite block of tokens removes a fraction of the current error. For direct learning alone, write

$$\ell_{t+1,p} = \ell_{t,p}\big(1 - \alpha(\Delta C_p)\big).$$

Approximating the error-reduction fraction locally by $\alpha(\Delta C_p) \approx \kappa \Delta C_p$ gives

$$\Delta\ell_p \approx -\kappa \ell_{t,p} \Delta C_p,$$

which motivates the round-wise rate $f_{t,p} = -\kappa\ell_{t,p}$. This is a local approximation used to rank allocations, not a claim that error decreases linearly over the full training run.

Other finite-step estimates of $f_{t,p}$ can be substituted. For example, a power-law model

$$\ell_p(C_p) = \ell_{\infty,p} + K_p(C_p + C_{0,p})^{-\beta_p}$$

gives the following average error change per token over a reference block of $h > 0$ tokens:

$$f_{t,p} = \frac{K_p}{h}\left[(C_{t,p} + h + C_{0,p})^{-\beta_p} - (C_{t,p} + C_{0,p})^{-\beta_p}\right].$$

Choose the reference block and compute these rates before selecting the next allocation. Holding $\mathbf{f}_t$ fixed during that round preserves the linear allocation model, score definition, and allocation results above.