Compositional Reasoning in Language Models under Reinforcement Learning Post-Training

Yu He
Stanford University
heyu@stanford.edu

Yingxi Li
Stanford University
yingxi@stanford.edu

Yifei Wang
Amazon AGI Labs
yifeiwg@amazon.com

Ellen Vitercik
Stanford University
vitercik@stanford.edu

Abstract

Compositional reasoning is critical for real-world problem solving: since training data is necessarily limited, models must generalize by composing learned skills in new ways. While post-training methods such as reinforcement learning (RL) have substantially improved the reasoning abilities of language models (LMs), their effects on compositional reasoning remain less well understood. We propose a dependency-graph framework to formalize compositional reasoning, yielding three levels of compositionality with increasing complexity. We conduct a systematic, controlled study of RL post-training by instantiating this framework with data-structure tasks, which provide deterministic reward computation and clear compositional structure. Extending prior findings in compositional generalization, we find that a decomposed-to-composed asymmetry persists under outcome-only RL on stateful tasks: decomposed-skill training does not reliably transfer to composed tasks, whereas composed-task training transfers more readily back to decomposed tasks. We provide theoretical explanation for this asymmetry, and further evaluate compositional generalization under length extrapolation, structural distribution shift, and transfer to tasks requiring unseen skills. Finally, we present a pilot study on tool-calling benchmarks, offering preliminary evidence that the decomposed-to-composed asymmetry may also arise in a practical setting.

1 Introduction

Compositional reasoning is a prerequisite for general intelligence. A model cannot merely learn individual skills; it must reuse and combine them systematically to solve unseen, more complex problems. This is necessary because real-world reasoning is open-ended, while training data is inherently limited. A model cannot be trained on every possible math problem, program, or tool-use trajectory it may encounter; instead, it must generalize by composing familiar skills in new ways.

As language models (LMs) take on increasingly complex reasoning tasks, understanding their compositional reasoning ability has become critical. However, existing studies of compositionality in LMs have largely focused on semantic composition in language understanding [30, 33] or instruction following [27]. In contrast, compositionality in reasoning tasks is less understood. Unlike linguistic composition, where primitives can cleanly correspond to words, phrases, or instructions, reasoning tasks often involve latent skills whose definitions and interactions are non-trivial to formalize. This makes it difficult to systematically evaluate whether models can compose reasoning skills.

On the other hand, reinforcement learning (RL) post-training has recently become a central approach for improving LM reasoning [6, 16, 23, 32]. Yet, most work on compositional reasoning studies

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

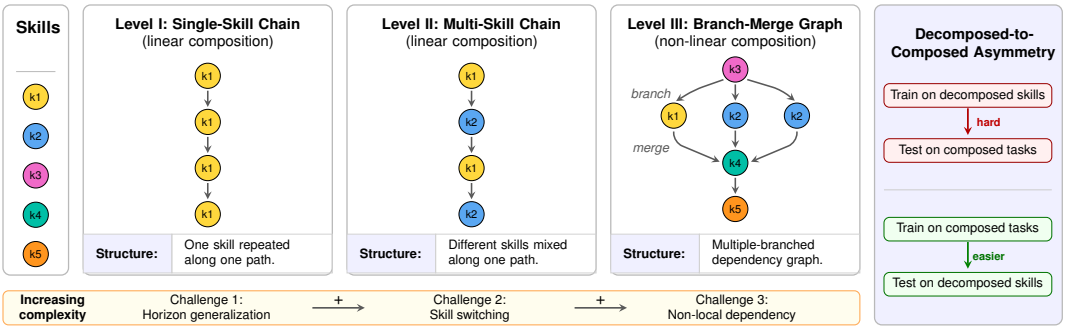


Figure 1: Three levels of compositionality with increasing complexity, shown as dependency structures among skills. Skills are defined as reusable primitives required to solve a reasoning task.

in-context learning [1, 2, 20, 28] or supervised finetuning [27, 29, 33]. Much less is known about the effects of RL on compositional reasoning. For example, one natural approach is to train on individual skills, where the smaller tasks may make it easier to design RL rewards and verifiers. However, it remains unclear whether LMs can compose the learned skills at test time.

To address this gap, we formalize compositional reasoning using a dependency-graph framework. We define skills as reusable primitives required to solve reasoning tasks, and use dependency graphs to model the compositional structure among them. This framework provides a formal basis for systematically evaluating compositional reasoning in language models.

For empirical evaluation, we instantiate the framework using data-structure tasks from DSR-Bench [10], a benchmark for evaluating structural reasoning in LMs. Data structures are well suited for this study because they provide deterministic, verifiable final outputs for RL reward computation. Moreover, these algorithmic tasks can be decomposed into clear skills, and the way these skills compose is explicit and unambiguous. For example, in binary-search-tree construction, insertion is the primitive skill, and construction requires applying this skill sequentially. This controlled setting allows us to compare decomposed-skill training with composed-task training and analyze how RL post-training affects the compositional reasoning ability of LMs.

A summary of our contributions:

- Our conceptual contribution is a framework formalizing compositional reasoning in LMs by representing tasks, where these skills and their dependencies can be identified, as dependency graphs over reusable primitive skills (Figure 1). This framework gives rise to three increasingly difficult levels of compositionality with distinct challenges: single-skill chains, which isolate horizon generalization; multi-skill chains, which add skill switching; and branch–merge graphs, which further add non-local dependencies across branches (Section 3).
- Our main empirical contribution is a systematic, controlled investigation of RL post-training on data-structure reasoning tasks (Section 4).[1] We compare RL decomposed-skill training and composed-task training across three compositionality levels and identify a consistent asymmetry: decomposed-skill training transfers poorly to composed tasks, whereas composed-task training transfers more readily back to decomposed tasks. These results extend prior findings in compositional generalization [27, 29] to outcome-based RL post-training on stateful reasoning tasks.
- We offer a theoretical explanation for the decomposed-to-composed asymmetry through error compounding and coverage shift, showing that composed reasoning requires sustained correctness over induced state distributions that decomposed supervision may not cover (Section 5).
- Finally, we conduct a small-scale, exploratory pilot study on a tool-calling benchmark, whose results are intended only as preliminary evidence that the decomposed-to-composed asymmetry may also arise in a practical setting (Section 6).

[1] Code can be found at `https://github.com/dransyhe/compositional-reasoning-rl`.

2 Related work

Compositional generalization and skill composition. Compositional generalization has been studied extensively in sequence-to-sequence learning, including SCAN [15], CFQ [11], and COGS [13], which evaluate whether models can recombine familiar primitives in novel semantic structures. Recent work extends this question to language models: SKILL-MIX [30] evaluates whether models can generate text combining multiple specified skills; Zhao et al. [33] study whether smaller models can learn higher-order skill combinations from lower-order examples; Yang et al. [27] study compositional generalization in instruction following; and Yin et al. [29] study composable chain-of-thought supervision for transferring from atomic to compositional tasks. These works study important forms of linguistic, instructional, or trace-level composition, while demonstrating difficulty of atomic to composition transfers. In contrast, we focus on stateful reasoning tasks, where the primitives and their interactions are often less explicit than surface-level linguistics, which were the primary tasks in prior works. Moreover, prior studies of atomic-to-composed transfer [27, 29] primarily rely on SFT, whereas we target outcome-based RL post-training.

RL post-training for compositional reasoning. RL post-training with verifiable rewards has recently become a central approach for improving language-model reasoning [6, 16, 23, 32], but its role in compositional transfer remains less understood. Closest to our work, Yuan et al. [31] show that RL can teach nested composition of string-transformation functions, Cheng et al. [4] teach atomic abilities through SFT and then apply compositional RL to knowledge-based tasks, while Xu et al. [26] propose Composition-RL, which constructs harder verifiable prompts by composing existing prompts for RL training. Li et al. [17] compare RL and SFT for cross-task and cross-modal composition in vision-language models. These works establish that RL can support composition or propose new training recipes, such as curriculum SFT-then-RL or compositional prompt construction. In contrast, our contribution is more diagnostic from a training-data perspective, specifically targeting stateful algorithmic reasoning tasks.

Structural and algorithmic reasoning. Dziri et al. [7] use computation graphs to analyze fixed algorithm executions and show that transformer performance can degrade with compositional complexity. DSR-Bench [10] evaluates structural reasoning through deterministic data-structure tasks, and recent graph-reasoning work uses RL to improve LMs on graph-theoretic tasks [9]. We build on this line of work by using data structures as a controlled environment for RL post-training, with deterministic, exact reward computation and clear compositional structure, while introducing a dependency-graph framework that characterizes how reusable skills compose and isolates distinct bottlenecks: horizon generalization, skill switching, and non-local dependencies.

3 A dependency-graph framework for compositionality

We present our conceptual contribution by formalizing compositional reasoning with a *dependency graph* over reusable skills (Figure 1), capturing compositional structures in reasoning tasks where the underlying skills and their dependencies can be identified. Our framework yields three hierarchical levels of compositionality that isolate distinct bottlenecks: horizon generalization, skill switching, and non-local dependencies. This structure also enables our analysis of decomposed-skill versus composed-task training. More broadly, it provides a formal basis for future empirical and theoretical studies of compositional reasoning in language models.

Dependency graph. For a task instance $\tau \in \mathcal{T}$, let the dependency graph $G_\tau = (V_\tau, E_\tau)$ be a finite directed acyclic graph (DAG), where each node represents a skill invocation and each edge indicates that the output of one invocation is required as input to another. Intuitively, G_τ specifies *which* intermediate computations must be performed, *which* skill is invoked at each step, and *how* they depend on one another; therefore, it represents the compositional structure within a task. We use a Binary Search Tree (BST) construction task over input $\{1, 2, 3\}$ as a running example.

- **Skill.** We assume a set of reusable skill primitives $\mathcal{K}$. Each skill type $k \in \mathcal{K}$ is associated with a function $f_k : \mathcal{X}_k \to \mathcal{Y}_k$, where $\mathcal{X}_k$ and $\mathcal{Y}_k$ denote the input and output spaces of that skill. For example, BST construction is composed of a sequence of `insert` operations, where `insert` acts as the reusable skill primitive.

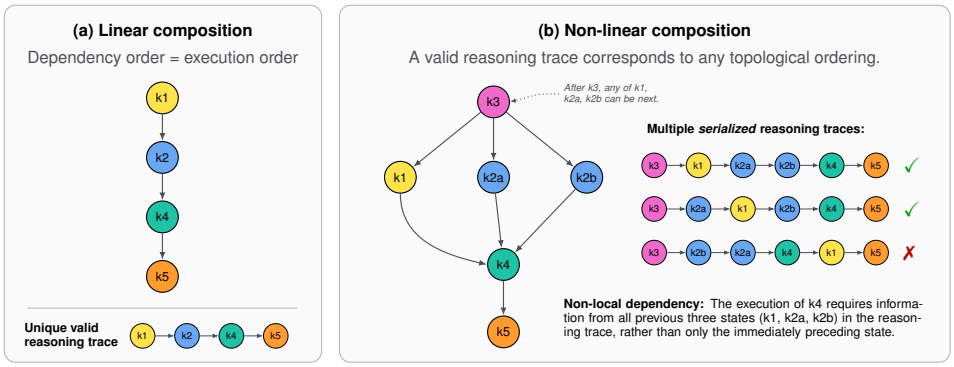


Figure 2: **Reasoning traces serialize dependency graphs.** An explicit step-by-step solution can be modeled as a topological ordering of the task dependency graph.

- **Node.** A node $v \in V_\tau$ is labeled by a skill type $\kappa(v) \in \mathcal{K}$, indicating which skill is invoked at that node. Let z_v denote the intermediate output produced by node v. For example, a concrete operation of `(insert, 1)` to the current BST is a skill invocation.
- **Edge.** An edge $(u, v) \in E_\tau$ indicates a dependency constraint that the intermediate output z_u is required to form the input to node v. Let $\mathrm{pred}(v) = \{u \in V_\tau : (u, v) \in E_\tau\}$ denote the predecessors of v. The input x_v and output z_v to node v are computed via

$$x_v = \psi_v\Big(\ell_v, (z_u)_{u \in \mathrm{pred}(v)}\Big) \in \mathcal{X}_{\kappa(v)}, \qquad z_v = f_{\kappa(v)}(x_v).$$

  Here, ℓ_v denotes local information specified by the task instance (e.g., what value to insert to the tree) and ψ_v is an input-construction map. For example, after `(insert, 1)`, the next operation `(insert, 2)` takes as input the BST state produced by the previous operation.
- **Output.** For simplicity, we assume that G_τ has a unique sink node $v_{\mathrm{sink}} \in V_\tau$. The final output of the task is the intermediate output $z_{v_{\mathrm{sink}}}$ produced at this sink node. For example, the final output of BST construction is given by the output of the skill invocation `(insert, 3)`.

Three levels of compositionality. As illustrated in Figure 1, we formalize three levels of compositionality under the dependency-graph framework.

- **Level I: Single-skill chain (linear composition).** Single-skill chain is the simplest setting, where the same skill is repeated across a sequence of steps. Formally, G_τ is a path, all nodes share the same skill type, and $|\{\kappa(v) : v \in V_\tau\}| = 1$. This setting tests whether a model trained on smaller instances can extrapolate to larger ones that require more repetitions of the same reasoning pattern.
- **Level II: Multi-skill chain (linear composition).** Multi-skill chain also has a path-structured dependency graph, but the skill label $\kappa(v)$ may vary across nodes with $|\{\kappa(v) : v \in V_\tau\}| > 1$. This setting tests whether a model can maintain intermediate state while sequentially composing heterogeneous skills, including handling skill switching and interface mismatches across skills.
- **Level III: Branch-merge graph (non-linear composition).** Here, G_τ is a general directed acyclic graph rather than a path. *Branching* occurs when one intermediate output feeds multiple downstream nodes, while *merging* occurs when a node depends on multiple upstream outputs.

Linear vs. non-linear compositions: non-local dependency. We model an explicit step-by-step solution as a sequential *reasoning trace* that serializes the dependency graph G_τ. Under this abstraction, any valid trace must respect the dependencies in G_τ, and thus corresponds to a topological ordering. As shown in Figure 2, linear compositions are chains, so the trace follows a unique order. In contrast, branch-merge graphs are non-linear: branches may be executed in different valid orders before being merged. At merge points, the model must retrieve and combine intermediate outputs from earlier branches, rather than only propagating the immediately preceding state, which therefore requires coordinating *non-local* dependencies.

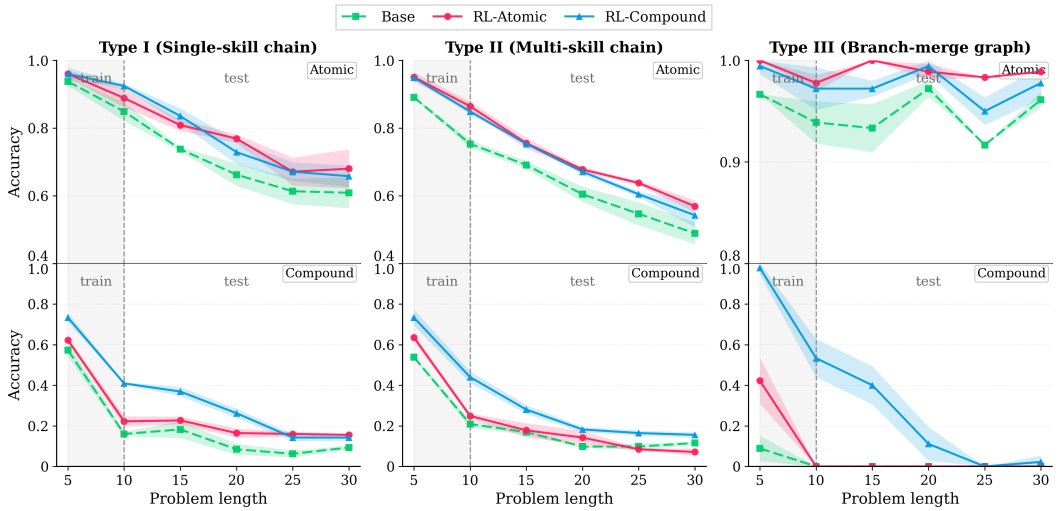


Figure 3: Evaluation on three compositionality types with Qwen3-4B-Instruct, averaged across three runs with the standard deviation shown as a shaded region. Rows evaluate *atomic* (decomposed skill) and *compound* (composed task) tasks. RL-Atomic trains on atomic tasks, while RL-Compound trains on compound tasks. Models are trained on small problem lengths (5-10) and evaluated on larger ones (up to 30). Results show a consistent decomposed-to-composed asymmetry.

4 Controlled RL study of compositional reasoning

We present our main empirical contribution: a systematic, controlled investigation of compositional reasoning under RL fine-tuning using the three-level framework above. We use data-structure tasks from DSR-Bench [10], which are well-suited because they provide deterministic, exact verification for RL reward computation, along with clear skill decompositions and compositional structure. We first present our main data-design finding, the decomposed-to-composed asymmetry. We then analyze the sources of compositional difficulty across the three levels and study generalization beyond length extrapolation.

4.1 Decomposed-skill vs. composed-task training

Our main experimental contrast compares RL post-training on decomposed skills and on composed tasks. We use five data-structure domains of varying difficulty: Array, Binary Search Tree (BST), Bloom Filter, Hashmap, and Heap. Task length denotes the initial input size for atomic tasks and the number of operations for compound tasks. Details on task description, example prompt, and data preparation can be found in Appendix C.

Atomic vs. compound tasks. An *atomic* task corresponds to a decomposed skill: a single skill-invocation applied to a given state. For example, in an atomic array task of length 3, given $[1, 2, 3]$, the model may be asked to insert 4 at position 2, yielding $[1, 2, 4, 3]$. A *compound* task corresponds to a composed task: a sequence of skill invocations, where intermediate states must be carried forward. For example, in a compound array task of length 3, given $[1, 2, 3]$, the model may execute "insert 4 at position 2, delete position 0, then insert 5 at position 1," and return the final array.

Decomposed-skill vs. composed-task training. We use Qwen3-4B-Instruct [19] as the main **Base** model, and additionally evaluate two more models from different families and sizes: OLMo-3-7B-Instruct [25] and Llama-3.1-8B-Instruct [8] on multi-skill chains. We use instruct models for their instruction-following ability, while excluding models already heavily post-trained for reasoning. For each base model, we compare **RL-Atomic**, trained only on atomic tasks, with **RL-Compound**, trained only on compound tasks. Both use Group Relative Policy Optimization [23] with binary final rewards, while keeping the number of training steps the same. Each problem in DSR-Bench has a uniquely verifiable final answer [10]. Refer to Appendix C for details on reward computation, training setup, and hyperparameters.

Main observation: Decomposed-to-composed asymmetry. As shown in Figures 3 and 4, we observe a consistent asymmetry across three levels of compositionality and model families. Atomic training often improves atomic-task performance, but does not reliably transfer to compound tasks. In contrast, compound training largely preserves or improves atomic-task performance while substantially improving compound-task performance. This suggests that learning decomposed skills in isolation is insufficient for learning how to compose them: composed-task training exposes the model to both the skills and their interactions. We provide theoretical support in Section 5. This shared observation motivates the remaining subsections, where we analyze sources of compositional challenges in a more fine-grained manner using the three-level framework introduced in Section 3.

4.2 Level I (Single-skill chain): Horizon generalization

Single-skill chained composition tests whether a model can repeatedly apply one skill invocation over a long dependent rollout. As shown in Figure 1, the dependency graph is a path with the same skill type at every node, yielding a unique valid serialized reasoning trace. This setting isolates horizon generalization, where horizon is the number of sequential skill invocations. Note this is distinct from problem length, which denotes input size. Empirically, we instantiate this setting by stacking insertion operations across the five data-structure domains, treating insertion as a skill.

Compositional challenge 1: horizon generalization. In Level I (Single-skill chain) of Figure 3, both RL-Atomic and RL-Compound consistently improve over the base model, indicating that RL post-training improves the underlying skill. However, the decomposed-to-composed asymmetry persists: RL-Atomic gives only marginal gains on compound tasks, whereas RL-Compound improves both atomic and compound performance. Moreover, performance drops much more sharply on compound than atomic tasks across all models, showing that even repeated invocation of a simple skill poses a horizon-generalization challenge: success requires staying correct across many intermediate states, where one early error can corrupt all later operations.

4.3 Level II (Multi-skill chain): Level I + Skill switching

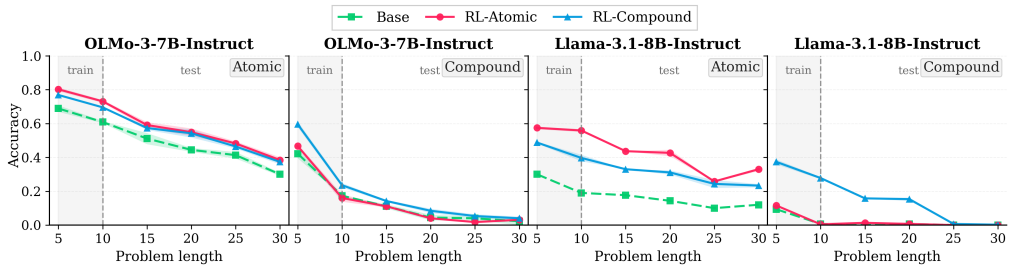


Figure 4: Multi-skill chained composition with OLMo-3-7B-Instruct and Llama-3.1-8B-Instruct.

Multi-skill chained composition adds a second bottleneck beyond horizon generalization: skill switching. Unlike single-skill chain, where one skill is repeatedly invoked, multi-skill chain requires composing heterogeneous skills in sequence and handling the interfaces between them, where the output produced by one skill must be converted into the appropriate input state for the next. We instantiate this setting by adding deletion as an additional skill across the five data-structure domains. Thus, RL-Atomic trains on single-step insertion or deletion, while RL-Compound trains on multi-step tasks with randomly interleaved insertions and deletions.

Compositional challenge 2: skill switching. The decomposed-to-composed asymmetry becomes more pronounced in Level II (Multi-skill chain) of Figure 3, and also appears for OLMo-3-7B-Instruct and Llama-3.1-8B-Instruct in Figure 4. With heterogeneous skill composition, RL-Compound clearly improves compound-task performance, whereas RL-Atomic provides little or no gain over the base models. On atomic tasks, RL-Compound remains comparable to RL-Atomic, indicating that compound training improves on both atomic and compound tasks. Figure 5 further shows that RL-Atomic stays close to the base model as the number of skill switches increases, suggesting that atomic training does not teach transitions between heterogeneous skills. Thus, decomposed-skill

training becomes insufficient beyond repeatedly stacking one skill: the model must learn to switch skills and propagate intermediate states across changing skill input and output requirements.

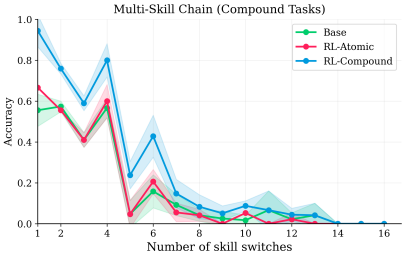


Figure 5: Accuracy as a function of the number of skill switches between insertion and deletion.

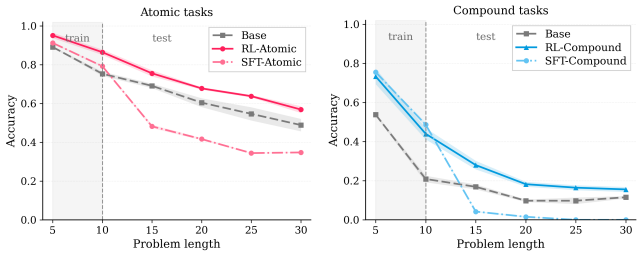


Figure 6: Comparison of post-training methods: supervised fine-tuning (SFT) vs. reinforcement learning (RL) for multi-skill chained composition.

SFT vs. RL post-training. Both supervised fine-tuning (SFT) and reinforcement learning (RL) are widely used post-training methods. We further compare RL with supervised fine-tuning (SFT) using programmatically generated SFT training samples. Data-structure tasks make this feasible because their algorithmic structure allows chain-of-thought templates for each skill to be instantiated with task-specific values. In principle, these structured traces should make the training pattern easy for SFT to learn. However, as shown in Figure 6, SFT performs well in-domain but degrades sharply on larger out-of-domain instances, whereas RL shows better length generalization. This suggests that SFT mainly learns the in-domain pattern, while RL better supports generalization beyond the training regime, consistent with prior findings [5]. Details of SFT training are in Appendix C.

4.4 Level III (Branch-merge graph): Level II + Non-local dependency

As contrasted in Section 3, unlike the linear compositions above, branch–merge graphs represent non-linear composition: computation branches into multiple intermediate chains and later merges their results. A valid reasoning trace may correspond to any topological ordering of the dependency graph, introducing a third bottleneck beyond horizon generalization and skill switching: non-local dependency coordination. At merge points, the model must retrieve, align, and combine outputs from earlier branches, rather than simply propagate the immediately preceding state. We use segment-tree construction as the branch–merge task, illustrated in Figure 7, with details deferred to Appendix C.

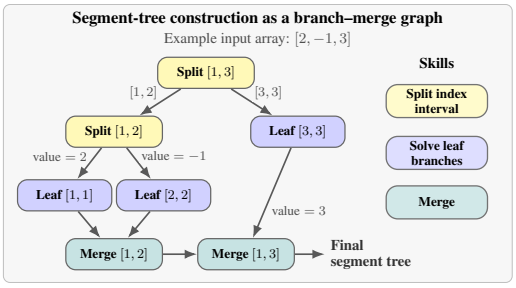


Figure 7: A simplified example illustration.

Compositional challenge 3: non-local dependency. The decomposed-to-composed asymmetry is most significant in Level III (Branch–merge graph) of Figure 3. All methods remain near ceiling on atomic tasks, indicating that the underlying skills are not difficult. However, a huge gap emerges on compound tasks: RL-Atomic gives little improvement over the base model and quickly collapses to near-zero accuracy, whereas RL-Compound performs substantially better. This suggests that decomposed-skill training is insufficient when reasoning requires coordinating non-local dependencies across branches. We note RL-Compound also degrades sharply as problem length increases, showing that non-local coordination is a stronger bottleneck than long horizon or skill switching alone.

4.5 Beyond length: structural shift and unseen skills

Above, we primarily vary problem length or execution horizon. We now test whether composed-task training generalizes under two additional forms of shift. First, Figure 8(a,b) tests robustness under structural distribution shift by varying BST skewness and the graph-construction threshold (which determines whether an edge is added between two nodes); composed-task training remains robust,

though its gains depend on structural difficulty. Second, Figure 8(c) tests transfer to tasks requiring unseen skills: multi-dimensional computation in KD-Tree and Geometric Graph construction, and mapping natural-language scenarios to formal BST and graph tasks in Clinical Appointments and Galaxy Traveling. Performance improves only modestly, suggesting that composed training alone is insufficient when new reasoning skills are required. Experiment details can be found in Appendix D.

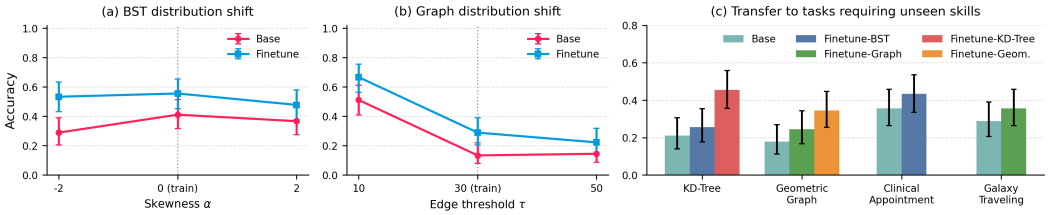


Figure 8: (a,b) Distribution-shift generalization on BST and graph construction. (c) Transfer to multi-dimensional and natural-language tasks. Error bars denote 95% confidence intervals.

5 Theoretical explanation of the asymmetry

We provide a theoretical explanation of the underlying mechanisms for the observed decomposed-to-composed asymmetry. Full statements, proofs, and additional analysis are deferred to Appendix B.

5.1 Why transfer from decomposition to composition is hard

Following the compositionality framework in Section 3, a task τ is represented by a dependency graph $G_\tau = (V_\tau, E_\tau)$. Each node $v \in V_\tau$ is a skill invocation with an input x_v and an output z_v. Atomic tasks involve only one skill invocation; *compound* tasks require multiple dependent invocations before returning the sink-node output $z_{v_{\text{sink}}}$.

We model an autoregressive solution as a serialized rollout of G_τ (Figure 2). For a *compound* task with horizon $|V_\tau| = T$, the step-t state s_t encodes the task, the executed-node set $P(s_t) \subset V_\tau$, and all intermediate outputs so far. The set of nodes that are ready to execute next is $R(s_t) = \{v \in V_\tau \setminus P(s_t) : \text{pred}(v) \subseteq P(s_t)\}$. Correctness is defined over a set of next invocation-output pairs because branch–merge graphs may admit several valid topological orderings. Define the *decision context* $Q(s_t) = \{(\kappa(v), x_v(s_t)) : v \in R(s_t)\}$, where $x_v(s_t)$ is the input to ready node v. Identifying invocations by skill type and input rather than node name, the set of valid invocation-output pairs is

$$\mathcal{I}^*(s_t) = \{((\kappa, x), f_\kappa(x)) : (\kappa, x) \in Q(s_t)\}.$$

Therefore, a policy $\hat{\pi}$ makes a mistake when $\hat{\pi}(s) \notin \mathcal{I}^*(s)$. Each valid step completes one ready node, so a rollout with no mistakes in T steps produces the correct final output. For stochastic policies, we assume the output distribution depends on the rollout history only through the current state; probabilities include policy randomness.

Let q be the decision-context distribution induced by the *atomic* task training data. Define the atomic-distribution error as $\epsilon_{\text{atom}}(\hat{\pi}; q) = \mathbb{E}_{c\sim q}[e_{\hat{\pi}}(c)]$.

For compound tasks $\tau \sim \rho_T$, let μ_t be the distribution of $Q(s_t)$ conditioned on there being no earlier mistakes. Assuming every context in the support of μ_t has nonzero probability under q for every $t \in [T]$, define the coverage factor

$$C_T(q) = \sup_{c:q(c)>0} \frac{\sum_{t=1}^{T} \mu_t(c)}{q(c)} < \infty,$$

which measures how well the training data covers contexts encountered across all steps of a compound rollout. It is large when a correct rollout contains contexts that are rare under atomic supervision.

Lemma 5.1 (informal)**.** *Under the preceding assumptions, let $F = 1$ denote the event that the model fails to produce the correct final output $z_{v_{\text{sink}}}$ on a compound task drawn from ρ_T. Then,*

$$\Pr[F = 1] \le C_T(q)\, \epsilon_{\text{atom}}(\hat{\pi}; q). \tag{1}$$

Moreover, this bound is tight in the worst case.

The bound in Equation 1 shows that decomposed-to-composed transfer requires more than low atomic error. The bound accumulates step errors across all T rollout steps, under decision-context distributions that atomic training may not cover well. Although the bound has no explicit multiplicative factor T, it is not horizon-free: whenever the policy can reach every rollout position with positive mistake-free probability, the cumulative coverage factor must satisfy $C_T(q) \geq T$. Thus, matching $q(c)$ to $T^{-1}\sum_{t=1}^{T}\mu_t(c)$ recovers an upper bound of $T\epsilon_{\text{atom}}(\hat{\pi}; q)$, while a poorer coverage weakens this guarantee. Proposition B.2 in Appendix B.1 shows that this horizon dependence is unavoidable.

Lemma 5.1 also helps interpret differences across the three levels of compositionality. Level I increases the rollout horizon, so even repeatedly applying a single skill leads to a $T\epsilon_{\text{atom}}(\hat{\pi}; q)$-scaling in the worst case. Level II introduces skill switches, so intermediate outputs must be carried across states around transitions between skill types. These transitions can produce skill-input pairs that are rare under q, increasing $C_T(q)$, consistent with Figure 5. Level III adds merge points whose correct execution depends on multiple predecessor outputs, and the resulting input tuples may be rare under q. Level III can also expose several distinct ready skill-input pairs simultaneously, creating non-singleton contexts that lie outside the support of any purely atomic training distribution. When reached with positive probability before the first mistake, these contexts create a structural gap in coverage: atomic error places no constraint on the policy's error at these decisions. Thus, even perfect atomic accuracy cannot rule out compound failure without additional assumptions linking performance on singleton and multi-invocation contexts.

5.2 Why transfer from composition to decomposition is easier

Performance generalization from compound to atomic tasks avoids accumulating errors across the rollout. Assume each compound task has a unique source node, so its initial decision context is a singleton. Therefore, under a non-recoverability assumption, good compound performance directly constrains the model's error on the first-step context induced by the compound-task distribution.

Let $\epsilon_{\text{comp}}(\hat{\pi}; \rho_T)$ be the compound-task error, and let s_1 be the initial state for $\tau \sim \rho_T$. The first-step error induced by the compound task distribution ρ_T is $\epsilon^{(1)}_{\text{atom}}(\hat{\pi}; \rho_T) = \Pr[\hat{\pi}(s_1) \notin \mathcal{I}^*(s_1)]$. If a mistake in the first step necessarily prevents successful completion, then $\epsilon^{(1)}_{\text{atom}}(\hat{\pi}; \rho_T) \leq \epsilon_{\text{comp}}(\hat{\pi}; \rho_T)$ because every first-step error is a compound failure.

For atomic test cases drawn from a target context distribution q covered by this first-step marginal, the same argument holds up to a coverage factor under context invariance. Let $q_1^{\rho_T}$ be the distribution over first-step decision contexts induced by compound tasks, i.e., $q_1^{\rho_T}(c) = \Pr_{\tau\sim\rho_T}[Q(s_1) = c]$. Assuming q puts no mass on contexts outside the support of $q_1^{\rho_T}$ and the coverage factor below is finite, we have

$$\epsilon_{\text{atom}}(\hat{\pi}; q) \leq C_1(q, \rho_T)\, \epsilon_{\text{comp}}(\hat{\pi}; \rho_T), \qquad \text{where } C_1(q, \rho_T) = \sup_{c: q_1^{\rho_T}(c) > 0} \frac{q(c)}{q_1^{\rho_T}(c)}.$$

When $q = q_1^{\rho_T}$, $C_1(q, \rho_T) = 1$, and small compound error implies small atomic error on this context distribution. Thus, under these assumptions, good compound performance controls error on covered atomic test cases, consistent with the RL-Compound results in Figures 3 and 4.

6 Pilot study: multi-call tool use

We conduct a small-scale pilot study to explore whether the decomposed-to-composed asymmetry may also arise in a more practical setting. We provide a pilot test using the Berkeley Function Calling Leaderboard (BFCL) [18], a benchmark that maps user requests to valid tool calls. Here, a skill corresponds to a tool-calling schema for a specific function. We define atomic tasks using the `simple_python` split, where each example requires one function call, and compound tasks using the `parallel` and `parallel_multiple` splits, where each request requires multiple calls. We use the same RL setup as

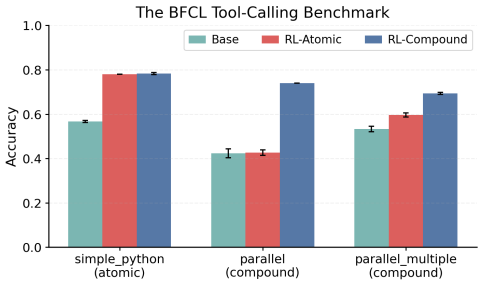


Figure 9: Decomposed-to-composed asymmetry in a tool-calling benchmark.

main experiments with Qwen3-4B-Instruct. Figure 9
shows a pattern consistent with the asymmetry observed in our controlled study: decomposed-skill training does not generalize well to compound tasks, whereas composed-task training performs well on both tasks. We note that this pilot study is intended to be preliminary and exploratory, rather than general evidence for real-world compositional reasoning. Experiment details are in Appendix C. We discuss potential implications of this design principle for proof writing, coding agents, and tool-calling agents in Appendix I.

7 Conclusion

We study compositional reasoning in language models under RL post-training, formalizing it as the execution of reusable skills over dependency graphs with three levels of compositional complexity. Using controlled data-structure tasks, we evaluate these levels and find that a decomposed-to-composed asymmetry, related to prior observations in compositional generalization, consistently arises under outcome-only RL on stateful reasoning tasks: training on decomposed skills transfers poorly to composed tasks, whereas training on composed tasks transfers more readily to decomposed settings. We explain this asymmetry through error compounding and coverage shift, and evaluate generalization beyond length extrapolation under structural distribution shifts and transfer to tasks requiring unseen skills. A pilot study in multi-call tool use provides preliminary evidence that the same asymmetry can arise in a practical domain. Overall, our results suggest that RL post-training cannot rely on decomposed skill supervision alone: exposing models to structured interactions among skills is important for improving composed reasoning while preserving performance on decomposed skills.

Limitations and future work. Our proposed framework applies to settings where skills and their dependencies are explicit, such as algorithmic tasks. Settings in which the underlying skills are latent or their composition is ambiguous are outside the scope, although we view our framework as a useful foundation for future work on latent skill discovery and composition in such less structured domains.

Empirically, our controlled study contrasts atomic training with compound training. Therefore, a more in-depth investigation of different mixtures of atomic and compound training data would help curate data recipes for RL post-training on compositional reasoning tasks. In addition, while we use GRPO as the primary training method, exploring a broader range of RL algorithms, along with reward shaping and curriculum designs, would provide a more comprehensive understanding of the topic. Further scaling experiments across different model sizes, particularly within the same model family, would show insights into scaling behavior. Finally, our empirical evaluation focuses primarily on controlled data-structure tasks; therefore, broader validation on real-world benchmarks remains an important direction for future work, with our BFCL pilot study as a first step.

Finally, we primarily evaluate compound tasks using final-output accuracy. Although intermediate-state accuracy could be informative, our preliminary studies found that requiring intermediate outputs reduced final accuracy, and potentially constrained models' ability to reason freely. Similarly, using Structured Outputs with JSON also reduced accuracy, likely due to added formatting burden. These design choices are further discussed in Appendix J. Future work could use LLM-based judging to track intermediate reasoning traces and parse final outputs, as well as using partial-credit rewards.

Contribution statement

YH led the project, including conceptual development, framework design, empirical studies, and paper writing. YL led the theoretical proofs on asymmetry. YW contributed to the conceptual framing and discussions. EV advised the project.

Acknowledgments and Disclosure of Funding

YH is supported by a Cubist PhD Fellowship. YL is supported by an Amazon AI Fellowship. EV gratefully acknowledges support from the National Science Foundation under award CCF-2338226 and the AI2050 program at Schmidt Sciences. We thank Connor Lawless, Anders Wikum, and Nikil Selvam for their feedback on this manuscript.

References

[1] Kartik Ahuja and Amin Mansouri. On provable length and compositional generalization, 2025. URL https://arxiv.org/abs/2402.04875.

[2] Hoyeon Chang, Jinho Park, Hanseul Cho, Sohee Yang, Miyoung Ko, Hyeonbin Hwang, Seungpil Won, Dohaeng Lee, Youbin Ahn, and Minjoon Seo. Characterizing pattern matching and its limits on compositional task structures. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=VCjlm003WL.

[3] Jiaao Chen, Xiaoman Pan, Dian Yu, Kaiqiang Song, Xiaoyang Wang, Dong Yu, and Jianshu Chen. Skills-in-context: Unlocking compositionality in large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 13838–13890, 2024.

[4] Sitao Cheng, Xunjian Yin, Ruiwen Zhou, Yuxuan Li, Xinyi Wang, Liangming Pan, William Yang Wang, and Victor Zhong. From atomic to composite: Reinforcement learning enables generalization in complementary reasoning. In *The 1st Workshop on Scaling Post-training for LLMs*, 2026. URL https://openreview.net/forum?id=CKijGdvOdd.

[5] Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. SFT memorizes, RL generalizes: A comparative study of foundation model post-training. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=dYur3yabMj.

[6] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

[7] 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 *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Fkckkr3ya8.

[8] Aaron Grattafiori et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

[9] Xiaojun Guo, Ang Li, Yifei Wang, Stefanie Jegelka, and Yisen Wang. G1: Teaching LLMs to reason on graphs with reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=Lq4nneD2xX.

[10] Yu He, Yingxi Li, Colin White, and Ellen Vitercik. Can LLMs reason structurally? benchmarking via the lens of data structures. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=kChzpoYFef.

[11] 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 *The Eighth International Conference on Learning Representations*, 2020.

[12] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*, 2023.

[13] Najoung Kim and Tal Linzen. COGS: A compositional generalization challenge based on semantic interpretation. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9087–9105, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.731. URL https://aclanthology.org/2020.emnlp-main.731/.

[14] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[15] Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In Jennifer Dy and Andreas Krause, editors, *Thirty-fifth International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2873–2882. PMLR, 10–15 Jul 2018.

[16] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Christopher Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=i1uGbfHHpH.

[17] Tianle Li, Jihai Zhang, Yongming Rao, and Yu Cheng. Unveiling the compositional ability gap in vision-language reasoning model. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=J76cCYTJub.

[18] Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=2GmDdhBdDk.

[19] Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[20] Rahul Ramesh, Ekdeep Singh Lubana, Mikail Khona, Robert P. Dick, and Hidenori Tanaka. Compositional capabilities of autoregressive transformers: A study on synthetic, interpretable tasks. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=L1eJ3NKPCd.

[21] Florian Redhardt, Yassir Akram, and Simon Schug. Scaling can lead to compositional generalization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=hZt0daVIZi.

[22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[23] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

[24] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys)*, 2025.

[25] Team Olmo, Allyson Ettinger, et al. Olmo 3, 2025. URL https://arxiv.org/abs/2512.13961.

[26] Xin Xu, Clive Bai, Kai Yang, Tianhao Chen, Yangkun Chen, Weijie Liu, Hao Chen, Yang Wang, Saiyong Yang, and Can Yang. Composition-rl: Compose your verifiable prompts for reinforcement learning of large language models. *arXiv preprint arXiv:2602.12036*, 2026.

[27] Haoran Yang, Hongyuan Lu, Wai Lam, and Deng Cai. Exploring compositional generalization of large language models. In Yang (Trista) Cao, Isabel Papadimitriou, Anaelia Ovalle, Marcos Zampieri, Francis Ferraro, and Swabha Swayamdipta, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 4: Student Research Workshop)*, pages 16–24, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-srw.3. URL https://aclanthology.org/2024.naacl-srw.3/.

[28] Jiaran Ye, Zijun Yao, Zhidian Huang, Liangming Pan, Jinxin Liu, Yushi Bai, Amy Xin, Liu Weichuan, Xiaoyin Che, Lei Hou, and Juanzi Li. How do transformers learn implicit reasoning? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=19ygs48nOa.

[29] Fangcong Yin, Zeyu Leo Liu, Liu Leqi, Xi Ye, and Greg Durrett. Learning composable chains-of-thought. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, 2026.

[30] Dingli Yu, Simran Kaur, Arushi Gupta, Jonah Brown-Cohen, Anirudh Goyal, and Sanjeev Arora. SKILL-MIX: a flexible and expandable family of evaluations for AI models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Jf5gplvglq.

[31] Lifan Yuan, Weize Chen, Yuchen Zhang, Ganqu Cui, Hanbin Wang, Ziming You, Ning Ding, Zhiyuan Liu, Maosong Sun, and Hao Peng. From $f(x)$ and $g(x)$ to $f(g(x))$: LLMs learn new skills in RL by composing old ones. In *The Fourteenth International Conference on Learning Representations*, 2026.

[32] Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=4OsgYD7em5.

[33] Haoyu Zhao, Simran Kaur, Dingli Yu, Anirudh Goyal, and Sanjeev Arora. Can models learn skill composition from examples? In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=1sLdprsbmk.

[34] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[35] Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V Le, Ed H. Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. Self-discover: Large language models self-compose reasoning structures. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Appendix

Table of Contents

A Additional related work **15**

B Omitted Proofs **15**

B.1 Omitted Proofs from Section 5.1 15

B.2 Omitted Proofs from Section 5.2 20

C Experiment details **22**

C.1 Dataset tasks 22

C.2 Example prompt 23

C.3 Data generation 24

C.4 Output validation 24

C.5 Reinforcement learning (RL) post-training 25

C.6 Supervised finetuning (SFT) post-training 25

C.7 The Berkeley Function Calling Leaderboard (BFCL) Benchmark 26

D Generalization of compositional reasoning **26**

D.1 Generalization under structural distribution shift 27

D.2 Generalization to unseen skills 27

E Ablation on mixing atomic and compound training data **28**

F Ablation on PPO **28**

G Ablation: additional model scale **29**

H Ablation: partial-credit reward **29**

I Practical implications **30**

J Structured output and intermediate scoring **30**

A Additional related work

Compositional generalization A line of work studies when and why compositional generalization emerges. Redhardt et al. [21] show that, with sufficient coverage, scaling data and model size can induce generalization to unseen task compositions, and provide theoretical results for modular task families. Ahuja and Mansouri [1] give guarantees for length and compositional generalization under sufficient training diversity. In controlled synthetic settings, Ramesh et al. [20] show that autoregressive transformers can generalize to many unseen compositions, while Chang et al. [2] characterize the limits of pattern matching on compositional task structures and show that coverage alone is not always sufficient. In symbolic reasoning, Ye et al. [28] study multi-hop generalization and find that atomic supervision mainly accelerates learning rather than determining it. However, these analyses are developed largely in supervised or tightly controlled settings, and do not directly address how post-training changes compositional reasoning in modern language models.

Compositional prompting. Related work proposes prompting methods that provide explicit test-time support for composition. Least-to-most prompting [34] asks the model to decompose a difficult problem into simpler subproblems and solve them sequentially. Decomposed prompting [12] delegates these subproblems to specialized prompt-based handlers, while Skills-in-Context prompting [3] places descriptions and demonstrations of foundational skills, along with examples of how they compose, in the prompt. Self-Discover [35] instead asks the model to select and combine general reasoning modules, such as critical thinking and step-by-step reasoning, into a task-specific reasoning structure. In contrast to these inference-time methods, we focus on a complementary question at a different stage of the pipeline: whether RL post-training, by improving atomic skills, can lead to improvements on composite tasks, and vice versa.

B Omitted Proofs

In this section, we provide omitted proofs and details from Section 5 of the paper.

B.1 Omitted Proofs from Section 5.1

The key challenge in transferring from decomposed skills to composed tasks is that solving a compound task requires the trained policy to remain correct over a serialized rollout of the task dependency graph. Even if each individual skill invocation is easy under the atomic training distribution, errors can accumulate over the reasoning trace, and the compound rollout may visit intermediate states that are rare under atomic supervision.

Formal rollout model. Fix a horizon T and let ρ_T be a distribution over compound task instances whose dependency graphs have $|V_\tau| = T$. A task instance $\tau \sim \rho_T$ is random, while T is fixed throughout this subsection. As in Section 3, write

$$G_\tau = (V_\tau, E_\tau)$$

for the dependency graph of τ. Each node $v \in V_\tau$ is labeled by a skill type $\kappa(v)$ and local information ℓ_v. Its input and correct output at a rollout state are defined below.

We write

$$\mathrm{pred}(v) = \{u \in V_\tau : (u, v) \in E_\tau\}$$

for the predecessors of v, and assume a unique sink node v_{sink} whose output is the final task output.

Let $\mathcal{S}$ be a countable rollout-state space. A state $s \in \mathcal{S}$ contains the task instance τ, the completed-node set

$$P(s) \subseteq V_\tau,$$

and the intermediate outputs produced so far, written as $\{z_u(s) : u \in P(s)\}$. Thus the state contains all task-relevant context needed to determine which skill invocations are available next and what their correct outputs are. The nodes that are ready to execute at state s are

$$R(s) = \{v \in V_\tau \setminus P(s) : \mathrm{pred}(v) \subseteq P(s)\}.$$

For each ready node $v \in R(s)$, its input is formed from the task-local information and predecessor outputs by

$$x_v(s) = \psi_v\Big(\ell_v, (z_u(s))_{u\in \mathrm{pred}(v)}\Big) \in \mathcal{X}_{\kappa(v)},$$

and its correct output is

$$z_v^*(s) = f_{\kappa(v)}(x_v(s)).$$

Decision context. A correct next step need not be unique, since a branch–merge graph may admit multiple valid topological orderings. Moreover, whether a step is correct does not depend on the name of the node being executed, but only on the skill type invoked and the input it receives. Therefore, for each state, we can define a decision context

$$Q(s) = \{(\kappa(v), x_v(s)) : v \in R(s)\},$$

which is the set of skill-input pairs available at state s. We then define the set of valid next invocation-output pairs by

$$\mathcal{I}^*(s) = \{((\kappa, x), f_\kappa(x)) : (\kappa, x) \in Q(s)\}.$$

By construction, $\mathcal{I}^*(s)$ is determined by $Q(s)$ only. Therefore, we write $\mathcal{I}^*(c)$ for a decision context c. Since $\mathcal{S}$ is countable, the set of decision contexts $\{Q(s) : s \in \mathcal{S}\}$ is also countable.

A rollout policy $\hat{\pi}$ maps a state s to a parsed invocation-output pair, consisting of the skill invocation it performs, identified by its skill type and input rather than by a node name, together with the output it produces. When $\hat{\pi}(s) = ((\kappa, x), z) \in \mathcal{I}^*(s)$, the rollout completes a ready node whose skill type is is κ and input is x, and records the correct output $z = f_\kappa(x)$. If several ready nodes have the same skill type and input, a fixed deterministic tie-breaking rule determines which one is completed, in accordance with our empirical deterministic tie-breaking rule in Appendix C.3. Note that when simultaneously ready nodes induce at least two distinct skill-input pairs, $Q(s)$ is non-singleton. For stochastic policies, all probabilities and expectations below are also over the policy randomness. We assume that, conditioned on the current rollout state s, the policy's current randomization is independent of the previous rollout history. Equivalently, any policy-internal memory or persistent random seed that can affect future outputs is included as part of the rollout state. Under this assumption, the stepwise error probability,

$$e_{\hat{\pi}}(s) = \Pr_{\hat{\pi}}[\hat{\pi}(s) \notin \mathcal{I}^*(s)],$$

is a well-defined function of state s. For a deterministic policy, the error probability is either 0 or 1, so $e_{\hat{\pi}}(s) = \mathbb{1}\{\hat{\pi}(s) \notin \mathcal{I}^*(s)\}$.

Context Invariance. We assume that the stepwise error probability depends on the state only through the decision context, i.e.,

$$Q(s) = Q(s') \Rightarrow e_{\hat{\pi}}(s) = e_{\hat{\pi}}(s'),$$

and we write $e_{\hat{\pi}}(c)$ for the stepwise error probability of a decision context c. This is because our transfer bound compares the policy on rollout distribution with the policy on atomic tasks, and this hypothesis is required to connect the two. By construction, the valid next-step set depends on the state only through its decision context: $Q(s) = Q(s')$ implies $\mathcal{I}^*(s) = \mathcal{I}^*(s')$. Thus, context invariance imposes no additional assumption on task correctness; it assumes only that the policy assigns the same total probability to invalid outputs at states with the same decision context. This assumption removes possible additional errors caused by embedding an invocation in a compound prompt, so the resulting transfer guarantee is optimistic.

Failure and mistake. Matching the main text, we write s_t for the rollout state at timestep t; although s_t is induced by the sampled task τ, we do not attach an additional τ superscript. The rollout starts from the initial state s_1, where $P(s_1) = \emptyset$, and then updates the completed-node set and intermediate outputs according to the parsed pair emitted by $\hat{\pi}$. When $\hat{\pi}(s_t) \in \mathcal{I}^*(s_t)$, the rollout completes one ready node with its correct output; otherwise, the next state may be arbitrary, but the step is counted as a mistake.

Let F denote the final-failure indicator:

$$F = \mathbb{1}\{\text{the rollout does not produce the correct sink-node output } z_{v_{\mathrm{sink}}}\}.$$

Equivalently, if the terminal rollout state stores the outputs produced by the serialized rollout, missing or incorrect sink-node output is counted as failure.

A mistake at step t is the event

$$\hat{\pi}(s_t) \notin \mathcal{I}^*(s_t).$$

Define the indicator of the first mistake made at step t by

$$M_t = \mathbb{1}\{\hat{\pi}(s_i) \in \mathcal{I}^*(s_i) \text{ for all } i < t, \quad \hat{\pi}(s_t) \notin \mathcal{I}^*(s_t)\}.$$

If no mistake occurs for $t = 1, \ldots, T$, then the rollout completes all nodes in a valid topological order and every completed node has the correct output. In particular, the sink-node output is $z_{v_{\text{sink}}}$, so $F = 0$. Hence $F = 1$ implies that $M_t = 1$ for at least one t.

Let q be the distribution over decision contexts induced by atomic task training data. Sample an atomic-task state s and record $Q(s)$. Define

$$\epsilon_{\text{atom}}(\hat{\pi}; q) = \mathbb{E}_{c \sim q}\left[e_{\hat{\pi}}(c)\right] = \sum_c q(c) e_{\hat{\pi}}(c).$$

When q is clear from context, we write $\epsilon_{\text{atom}}(\hat{\pi})$. An atomic task instance has a single-node dependency graph, so its decision context is a singleton $\{(\kappa, x)\}$, and q is supported on singleton contexts. A compound rollout context can therefore be covered by q only if it is the same singleton and has positive q-mass. This singleton condition holds structurally along Level I and Level II paths, but may fail in Level III when several distinct invocations are simultaneously available. We compare atomic and compound behavior at the level of decision contexts because this captures their possible overlap; it does not assume that every compound decision context has an atomic analogue.

We demonstrate the notation above with a concrete example.

Example B.1 (BST construction)**.** Consider the following prompt:

```
 You are given an initially empty binary search tree.  Execute
the following operations in order:
1.  insert 5
2.  insert 2
3.  insert 8
4.  delete 2
5.  insert 3
Use [] for an empty child and [v,L,R] for a nonempty node with
value v, left subtree L, and right subtree R. Return only the
final tree.
```

This prompt is a task instance τ with horizon $T = 5$. Its dependency graph is a path $v_1 \to v_2 \to v_3 \to v_4 \to v_5$, where each node is one BST skill invocation. The local information ℓ_{v_i} specifies the instructed operation, such as `insert 5` for v_1 and `delete 2` for v_4. The sink node is $v_{\text{sink}} = v_5$, and the correct sink-node output is

$$z_{v_{\text{sink}}} = [5, [3, [], []], [8, [], []]].$$

Let B_t denote the BST after the first $t - 1$ invocations have been correctly executed. Along the correct rollout,

$$B_1 = [], \quad B_2 = [5, [], []], \quad B_3 = [5, [2, [], []], []], \quad B_4 = [5, [2, [], []], [8, [], []]],$$

$$B_5 = [5, [], [8, [], []]], \quad B_6 = [5, [3, [], []], [8, [], []]].$$

At timestep t, the rollout state can be represented as

$$s_t = (\tau, \{v_1, \ldots, v_{t-1}\}, \{z_{v_i} = B_{i+1} : i < t\}).$$

Because the graph is a path, the ready-node set is a singleton at every nonterminal correct state: $R(s_t) = \{v_t\}$. For example, at step 4,

$$P(s_4) = \{v_1, v_2, v_3\}, \qquad R(s_4) = \{v_4\}.$$

so the decision context and the valid next invocation-output set are the singletons

$$Q(s_4) = \{(\texttt{delete}, (B_4, 2))\}, \qquad \mathcal{I}^*(s_4) = \{((\texttt{delete}, (B_4, 2)), B_5)\}.$$

If the model invokes the available skill-input pair but returns an incorrect intermediate BST, or proposes a skill-input pair not in $Q(s_4)$, such as insert 3 before delete 2 is performed, then its invocation-output pair does not belong to $\mathcal{I}^*(s_4)$.

The atomic task instance "given the tree B_4, delete 2" has the same decision context $Q(s_4)$, even though its dependency graph has one node while that of τ has five. Decision contexts are exactly the level of description at which an atomic training example and a step of a compound rollout can coincide.

Lemma 5.1 shows that generalizing from atomic to compound tasks depends on both the atomic error on q and the mismatch between q and the decision contexts visited along mistake-free compound rollouts.

Lemma 5.1. *Assume $\hat{\pi}$ is context-invariant. For each step t, let μ_t be the conditional distribution of the decision context $Q(s_t)$ given no mistakes before step t:*

$$\mu_t(A) = \Pr[Q(s_t) \in A \mid M_1 = \cdots = M_{t-1} = 0] \qquad \textit{for a set } A \textit{ of decision contexts.}$$

Assume that $q(c) = 0 \Rightarrow \mu_t(c) = 0$ for every $t \in [T]$ and every decision context c, i.e., any decision context that can occur at timestep t of a correct compound rollout has nonzero probability under the atomic training-state distribution q. Define the cumulative coverage factor

$$C_T(q) = \sup_{c:q(c)>0} \frac{\sum_{t=1}^{T} \mu_t(c)}{q(c)}.$$

Assume $C_T(q) < \infty$. Then, for $\tau \sim \rho_T$, the probability of final failure satisfies

$$\Pr[F = 1] \le C_T(q)\, \epsilon_{\mathrm{atom}}(\hat{\pi}; q).$$

This bound is tight in the worst case, as shown by Proposition B.2.

Proof. For each t, define the conditional step-error probability

$$\delta_t(\hat{\pi}) = \Pr[\hat{\pi}(s_t) \notin \mathcal{I}^*(s_t) \mid M_1 = \cdots = M_{t-1} = 0].$$

As argued above, if the final output is incorrect, then at least one first mistake must have occurred. Hence

$$\Pr[F = 1] \le \sum_{t=1}^{T} \Pr[M_t = 1] = \sum_{t=1}^{T} \Pr[M_1 = \cdots = M_{t-1} = 0]\, \delta_t(\hat{\pi}) \le \sum_{t=1}^{T} \delta_t(\hat{\pi}).$$

Fix t. Conditioned on the rollout state $s_t = s$, the policy makes a mistake with probability $e_{\hat{\pi}}(s)$, which by context invariance equals $e_{\hat{\pi}}(Q(s))$. Conditioning instead on the decision context and using that the set of decision contexts is countable,

$$\delta_t(\hat{\pi}) = \sum_c \mu_t(c)\, e_{\hat{\pi}}(c).$$

Summing over t and using the coverage assumption, which allows us to restrict the sum to contexts with $q(c) > 0$,

$$\begin{aligned}\sum_{t=1}^{T} \delta_t(\hat{\pi}) &= \sum_c \left(\sum_{t=1}^{T} \mu_t(c) \right) e_{\hat{\pi}}(c) = \sum_{c:q(c)>0} \frac{\sum_{t=1}^{T} \mu_t(c)}{q(c)}\, q(c)\, e_{\hat{\pi}}(c) \\ &\le C_T(q) \sum_c q(c)\, e_{\hat{\pi}}(c) = C_T(q)\, \epsilon_{\mathrm{atom}}(\hat{\pi}; q).\end{aligned}$$

Combining the preceding inequalities proves the claim.

□

Lemma 5.1 shows that transferring from atomic supervision to compound tasks depends on two quantities: the atomic error $\epsilon_{\mathrm{atom}}(\hat{\pi}; q)$ and the coverage factor $C_T(q)$, which measure the mismatch between the atomic training-state distribution and the rollout state distribution. This guarantee is not horizon-free: the horizon dependence is absorbed into the coverage factors $C_T(q)$.

Indeed, suppose $\Pr[M_1 = \cdots = M_{t-1} = 0] > 0$ for every $t \in [T]$, so that each μ_t is a probability distribution and $\sum_c \sum_{t=1}^{T} \mu_t(c) = T$. Restricting to contexts with $q(c) > 0$, which the coverage assumption permits,

$$T = \sum_{c:q(c)>0} \frac{\sum_{t=1}^{T} \mu_t(c)}{q(c)}\, q(c) \le C_T(q) \sum_{c:q(c)>0} q(c) = C_T(q).$$

So $C_T(q) \ge T$ always, with equality exactly when atomic supervision matches the visitation frequencies of the compound rollout, i.e. $\sum_t \mu_t = Tq$. In particular, even perfectly matched coverage still pays a factor T, recovering the $T\epsilon_{\mathrm{atom}}$-scaling; poorer coverage further amplifies failure risk. Note that the coverage assumption assumes atomic supervision is capable of producing the contexts a compound rollout visits. This holds for path-structured graphs, where $|R(s_t)| = 1$ along a correct rollout so every visited context is a singleton, but not in general; we return to this point in the discussion of Level III below.

The following proposition shows that this horizon dependence is unavoidable even in a path-structured, unique-valid-invocation setting.

Proposition B.2 (No horizon-free transfer bound)**.** *For every $T \ge 1$, there exist a compound task distribution ρ_T, a countable state space $\mathcal{S}$, an atomic training distribution q over decision contexts, and a context-invariant policy $\hat{\pi}$ such that each non-terminal state on the mistake-free rollout has a singleton valid next invocation-output set, yet*

$$\epsilon_{\mathrm{atom}}(\hat{\pi}; q) = \frac{1}{T}, \qquad \Pr[F = 1] = 1.$$

Consequently, no bound of the form

$$\Pr[F = 1] \le g(\epsilon_{\mathrm{atom}}(\hat{\pi}; q))$$

with $g(\epsilon) \to 0$ as $\epsilon \to 0$ can hold uniformly over all horizons T and all compound task distributions.

Proof. Let ρ_T be the point mass on the following single-skill chain: starting from the empty list, insert the values $1, 2, \ldots, T$ in this order. Its dependency graph is a path of length T, so the ready-node set is a singleton at every nonterminal correct state, and the correct output of each invocation is unique. Along the correct rollout, the decision context at step t is the singleton

$$c_t = \{(\texttt{insert}, ([1, \ldots, t-1], t))\},$$

and $c_1, \ldots, c_T$ are pairwise distinct because their inputs are. Each c_t is also the decision context of an atomic task instance, namely the single invocation `insert` t applied to the list $[1, \ldots, t-1]$, so the uniform distribution q over $\{c_1, \ldots, c_T\}$ is realizable as an atomic training distribution.

Let $\hat{\pi}$ be the deterministic policy, defined on decision contexts and hence context-invariant, that returns the valid pair on $c_1, \ldots, c_{T-1}$ and a pair outside $\mathcal{I}^*(c_T)$ on c_T. As the rollout model permits, let a mistake send the rollout to a state from which the correct sink-node output is never produced. Then $\hat{\pi}$ errs on exactly one of the T contexts in the support of q, so

$$\epsilon_{\mathrm{atom}}(\hat{\pi}; q) = \frac{1}{T}.$$

However, the compound rollout makes no mistake at steps $1, \ldots, T-1$ and therefore visits $c_1, c_2, \ldots, c_T$ in order, necessarily making a mistake at step T. Thus, the rollout fails to produce the correct sink-node output, so

$$\Pr[F = 1] = 1.$$

□

In this construction, μ_t is the point mass on c_t, so $\sum_{t=1}^{T} \mu_t(c_j) = 1$ for any $j \in [T]$, while $q(c_j) = 1/T$; hence $C_T(q) = T$ and the bound of Lemma 5.1 evaluates to $T \cdot \frac{1}{T} = 1 = \Pr[F = 1]$. Thus Lemma 5.1 is tight in the worst case.

Connection with the compositionality levels. Lemma 5.1 explains how the difficulty of composing atomic tasks changes across our three levels of compositionality.

- **Level I (Single-skill chain)**: In this case, G_τ is a path, and all nodes share the same skill type, so the main challenge is the rollout horizon $T = |V_\tau|$ (depth generalization). Even without skill switches or merges, a length-T rollout incurs T context visits. The cumulative factor counts all of these visits, including repeated visits to the same context, and therefore satisfies $C_T(q) \geq T$ under the stated conditions, with equality only under perfectly matched coverage. For the natural uniform mixture over visited contexts, this recovers the familiar $T\epsilon_{\text{atom}}$-type scaling.
- **Level II (Multi-skill chain)**: In this case, the skill invoked at each node $v \in V_\tau$, namely $\kappa(v)$, varies across nodes. The model must therefore not only remain correct over a long horizon, but also execute different skills in sequence and propagate intermediate outputs across skill switches. Even if each skill is well covered in isolation by atomic supervision, the compound rollout distribution near a skill switch may assign mass to contexts in which one skill is applied to an input produced by a different skill, and such cross-skill contexts may receive little mass under the atomic training distribution q, increasing $C_T(q)$. This matches our experimental results in Figure 5: as the number of skill switches increases, the task becomes more difficult for the model.
- **Level III (Branch-merge graph)**: In this case, G_τ is a branch–merge graph, and two effects arise that have no counterpart in Levels I and II. First, whenever simultaneously ready branches induce at least two distinct skill-input pairs, $Q(s_t)$ contains more than one invocation. The model must also choose which of them to execute next, a decision with no atomic analogue. Since an atomic task instance has a single-node dependency graph, its decision context is a singleton, so every such multi-invocation context lies outside the support of any atomic training distribution q. The coverage assumption of Lemma 5.1 then fails unless the training distribution is enlarged beyond single-node atomic tasks to include multi-invocation choice contexts. Second, a merge invocation is a single skill applied to a tuple of outputs from different branches, and so is atomically realizable, but the contexts that matter are those in which the tuple was produced by two separately executed branches; if atomic supervision covers merge invocations only on tuples from short or isolated branches, those contexts receive little mass and $C_T(q)$ is large. Both effects are consistent with the collapse of RL-Atomic on Level III compound tasks.

B.2 Omitted Proofs from Section 5.2

The previous subsection shows that atomic-to-compound transfer is difficult because errors can accumulate over a multi-step rollout. The reverse direction does not have the same horizon dependence: an error at the first step of a compound rollout is already an error on a single skill invocation.

Let ρ_T denote a distribution over compound task instances with horizon T. For $\tau \sim \rho_T$, let s_1 be its initial rollout state. Define the compound-task error by

$$\epsilon_{\text{comp}}(\hat{\pi}; \rho_T) = \mathbb{E}_{\tau \sim \rho_T, \hat{\pi}}[F],$$

and define the first-step error induced by ρ_T as

$$\epsilon^{(1)}_{\text{atom}}(\hat{\pi}; \rho_T) = \Pr_{\tau \sim \rho_T, \hat{\pi}}\left[\hat{\pi}(s_1) \notin \mathcal{I}^*(s_1)\right].$$

To interpret this first-step error as an error with atomic context, we assume that every compound task in the support of ρ_T has a unique source node. Since the completed-node set $P(s_1) = \emptyset$, the ready node set $R(s_1)$ contains exactly one node, and the decision context $Q(s_1)$ is a singleton decision context.

Proposition B.3 (First-step compound-to-atomic transfer)**.** *Assume that a first-step mistake necessarily prevents successful completion of the compound task, i.e., for every task instance τ and every realization of the policy randomness,*

$$\hat{\pi}(s_1) \notin \mathcal{I}^*(s_1) \quad \Longrightarrow \quad F = 1.$$

Then

$$\epsilon^{(1)}_{\text{atom}}(\hat{\pi}; \rho_T) \leq \epsilon_{\text{comp}}(\hat{\pi}; \rho_T).$$

Proof. Recall that $M_1 = \mathbb{1}\{\hat{\pi}(s_1) \notin \mathcal{I}^*(s_1)\}$ is the indicator of whether the model makes mistake on the first step. By the non-recoverable mistake assumption, $M_1 \leq F$ for every task instance and

every realization of the policy randomness. Taking the expectation gives $\mathbb{E}[M_1] \leq \mathbb{E}[F]$, and notice that the left-hand side is $\epsilon^{(1)}_{\text{atom}}(\hat{\pi}; \rho_T)$ and the right-hand side is $\epsilon_{\text{comp}}(\hat{\pi}; \rho_T)$. □

Proposition B.3 compares two events within the same compound rollout, so no coverage assumption or context invariance assumption is required here.

Transfer to a separate atomic distribution. Even when an atomic task and the first step of a compound task require the same skill on the same input, their full rollout states differ because each state records the entire task instance. We therefore compare them through their decision contexts.

Define the first-step decision-context distribution induced by compound tasks as

$$q_1^{\rho_T}(c) = \Pr_{\tau \sim \rho_T} [Q(s_1) = c] .$$

Because every task has a unique source node, $q_1^{\rho_T}$ is supported on singleton decision context.

Assume that $\hat{\pi}$ is context invariant. Then the first-step error can be written as

$$\begin{aligned} \epsilon^{(1)}_{\text{atom}}(\hat{\pi}; \rho_T) &= \mathbb{E}_{\tau \sim \rho_T} [e_{\hat{\pi}}(s_1)] \\ &= \mathbb{E}_{\tau \sim \rho_T} [e_{\hat{\pi}}(Q(s_1))] \\ &= \sum_c q_1^{\rho_T}(c) e_{\hat{\pi}}(c). \end{aligned}$$

Now let q be a target atomic distribution over decision contexts. Assume that q is covered by the compound first-step distribution:

$$q_1^{\rho_T}(c) = 0 \quad \Longrightarrow \quad q(c) = 0.$$

Define the first-step coverage factor as

$$C_1(q, \rho_T) = \sup_{c : q_1^{\rho_T}(c) > 0} \frac{q(c)}{q_1^{\rho_T}(c)}.$$

Then

$$\epsilon_{\text{atom}}(\hat{\pi}; q) \leq C_1(q, \rho_T)\, \epsilon_{\text{comp}}(\hat{\pi}; \rho_T).$$

Indeed, using the definition of atomic error,

$$\begin{aligned} \epsilon_{\text{atom}}(\hat{\pi}; q) &= \sum_c q(c) e_{\hat{\pi}}(c) \\ &= \sum_{c : q_1^{\rho_T}(c) > 0} \frac{q(c)}{q_1^{\rho_T}(c)} q_1^{\rho_T}(c) e_{\hat{\pi}}(c) \\ &\leq C_1(q, \rho_T) \sum_c q_1^{\rho_T}(c) e_{\hat{\pi}}(c) \\ &= C_1(q, \rho_T)\, \epsilon^{(1)}_{\text{atom}}(\hat{\pi}; \rho_T) \\ &\leq C_1(q, \rho_T)\, \epsilon_{\text{comp}}(\hat{\pi}; \rho_T), \end{aligned}$$

where the final inequality follows from Proposition B.3.

When $q = q_1^{\rho_T}$, we have $C_1(q, \rho_T) = 1$, and therefore

$$\epsilon_{\text{atom}}(\hat{\pi}; q_1^{\rho_T}) \leq \epsilon_{\text{comp}}(\hat{\pi}; \rho_T).$$

Unlike atomic-to-compound transfer, this bound has no horizon factor: it compares atomic performance with only the first decision context of the compound rollout, rather than accumulating errors over all T steps.

C Experiment details

C.1 Dataset tasks

Five data structure domains. We chose five common data structure domains of varying difficulty, taken from DSR-Bench [10]: array, binary search tree, bloom filter, hashmap, and heap.

- **Array.** An array is an ordered list of values indexed from left to right. The model inserts a value at an index or deletes an element, and outputs the final array as a list of integers.
- **Binary Search Tree (BST).** A BST is a binary tree where values in the left subtree are smaller than the node and values in the right subtree are larger. The tree is described through node relationships, and the output is the pre-order traversal of the final tree as a flattened list.
- **Bloom Filter.** We use a counting Bloom filter, which represents a set with an array of counters and several hash functions. Insert increments the counters at the hashed positions, while delete decrements them without going below zero. The output is the final count array.
- **Hashmap.** A hashmap stores key-value pairs in buckets using a hash function. Insert adds or updates a key-value pair, and delete removes a key if it exists. The output is a nested list of buckets, where each bucket contains its key-value pairs.
- **Heap.** We use an array-based min-heap, where each parent is less than or equal to its children. Insert adds a value and restores the heap property by sifting up, while delete removes the root and restores the heap by sifting down. The output is the final heap array.

The atomic skills are INSERT and DELETE for each domain, with deletion called REMOVE for BSTs and hashmaps in the implementation. The compound task combines these skills: each example gives a sequence of insert and delete operations, and the model must apply them in order and return the final data structure state.

The problem length is defined as the initial size of the input data structure, such as the number of elements in an array, nodes in a tree, or entries in a hashmap. For compound tasks, the problem length additionally also indicates the number of operations in the chain, since the model must update the data structure repeatedly before producing the final state.

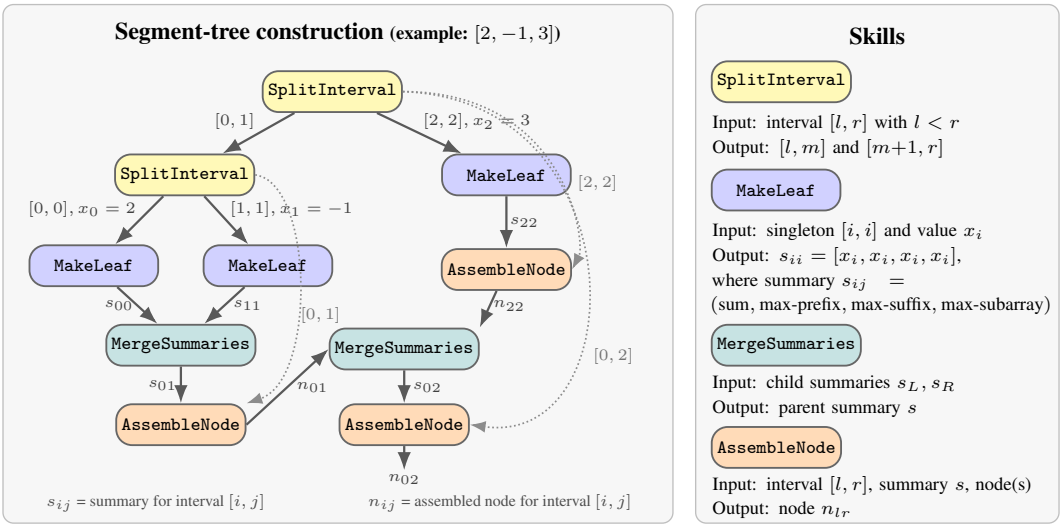


Figure 10: An example of segment-tree construction as branch-merge graph.

Segment tree construction. Given an integer array, the model must construct a segment tree over the full index range $[0, n-1]$. A segment tree is a binary tree that recursively partitions an array interval into smaller intervals. In our setting, every node corresponds to an inclusive interval $[l, r]$ and stores a four-field summary

$$[\texttt{sum}, \texttt{best_prefix}, \texttt{best_suffix}, \texttt{best_subarray}].$$

Here, sum is the total sum of the interval, best_prefix is the maximum sum of a non-empty prefix subarray, best_suffix is the maximum sum of a non-empty suffix subarray, and best_subarray is the maximum sum of any non-empty contiguous subarray inside the interval. Thus, the constructed tree represents both the recursive interval structure and the dynamic-programming summaries needed for maximum-subarray range queries.

We decompose segment-tree construction into four skills:

- SplitInterval: Given an interval $[l, r]$ with $l < r$, compute $m = \lfloor (l + r)/2 \rfloor$ and return the two child intervals $[l, m]$ and $[m + 1, r]$.
- MakeLeaf: Given a single array value x, construct the leaf summary $[x, x, x, x]$, since the sum, best prefix, best suffix, and best subarray are all equal to the only element.
- MergeSummaries: Given left summary $[L_{\text{sum}}, L_{\text{pref}}, L_{\text{suff}}, L_{\text{best}}]$ and right summary $[R_{\text{sum}}, R_{\text{pref}}, R_{\text{suff}}, R_{\text{best}}]$, compute the parent summary:

$$\begin{aligned} \texttt{sum} &= L_{\text{sum}} + R_{\text{sum}}, \\ \texttt{best_prefix} &= \max(L_{\text{pref}}, L_{\text{sum}} + R_{\text{pref}}), \\ \texttt{best_suffix} &= \max(R_{\text{suff}}, R_{\text{sum}} + L_{\text{suff}}), \\ \texttt{best_subarray} &= \max(L_{\text{best}}, R_{\text{best}}, L_{\text{suff}} + R_{\text{pref}}). \end{aligned}$$

- AssembleNode: Package an interval, its summary, and its two children into the node representation $[\texttt{interval}, \texttt{summary}, \texttt{left}, \texttt{right}]$. Leaf children are represented as None.

The compound construction task applies these skills recursively. Starting from $[0, n - 1]$, if $l = r$, the model applies MakeLeaf and then AssembleNode. If $l < r$, the model applies SplitInterval, independently constructs the left and right subtrees, applies MergeSummaries to combine their summaries, and finally applies AssembleNode to form the parent. Branching occurs at each non-leaf interval when it is split into two child intervals that are processed independently. Merging occurs after both child subtrees are complete, when their summaries are combined into the parent summary.

The final answer is a pre-order traversal of the constructed tree. Each visited node is represented as

$$[[l, r], [\texttt{sum}, \texttt{best_prefix}, \texttt{best_suffix}, \texttt{best_subarray}]].$$

C.2 Example prompt

Below is an example array-compound task of length 6:

```
An array supports the following operations (using 0-based indexing):
1. (insert, index=i, value=v) inserts value v at position i,
   shifting elements at i and beyond to the right.
2. (delete, index=i) removes the element currently at position i.
Start with the provided initial array and apply each operation in order.

Initial array: [94, 89, 79, 60, 41, 48]
Operations:
(delete, index=4)
(delete, index=4)
(insert, index=1, value=33)
(insert, index=4, value=26)
(delete, index=3)
(delete, index=0)

Q: What is the final array after applying all operations?
Your answer should be a list of integers, e.g. [1, 2, 3].

Approach the problem methodically. Ensure all conclusions are based on
precise calculations and logical deductions. Feel free to explore various
solution methods and cross-check results for consistency. Maintain dynamic
thinking and always verify each step of your reasoning. The last line of
your response should be of the following format: 'Therefore, the final
answer is: $\boxed{ANSWER}$.' (without quotes), where ANSWER is just the
final number or expression that solves the problem. Think carefully and
break down the problem step by step.
```

We append the instruction "Approach the problem methodically...". This instruction follows Guo et al. [9], which prompts the model to think carefully step by step before giving the final answer.

C.3 Data generation

All examples are synthetically and programmatically generated. This is possible because the behavior of each data structure can be implemented exactly in code, which lets us compute the ground-truth answer without human annotation. Each data structure has a prompt template describing its rules and output format (an example see above). We then populate the template with randomly generated values, using either integers (in the range 0–100) or strings over the English alphabet, depending on the task domain.

For multi-skill chain's compound tasks, each example starts from an initial data structure state and applies a chain of operations. Each operation is sampled as an insert with probability 0.7 and a delete with probability 0.3. We ensure that every deletion is valid: for example, array deletion chooses an existing index, BST and hashmap deletion choose existing keys or values, heap deletion is only applied to a non-empty heap, and counting Bloom filter deletion chooses an item that has previously been inserted. The answer is the final serialized state after applying all operations in order.

We also make the outputs deterministic and unique by fixing implementation details that could otherwise be ambiguous. For example, when a traversal has multiple possible next nodes, we visit neighbors in increasing value order; when heap deletion has equal children, we use a fixed tie-breaking rule. These constraints ensure that each prompt has a single well-defined ground-truth output.

C.4 Output validation

We extract the model's final answer from the final `\boxed{}` answer and compare it against the ground truth, following Guo et al. [9]. We use instruction-tuned models to reduce errors caused by poor instruction following rather than poor data-structure reasoning. To further reduce formatting sensitivity, our evaluator applies several formatting relaxations:

- it extracts the last boxed answer, including boxed answers with nested braces;
- it accepts `\boxed` answers written without braces;
- it falls back to answers following phrases such as "the final answer is";
- it strips common wrappers such as code fences and simple LaTeX text wrappers;
- it extracts the first balanced list, tuple, or dictionary from otherwise verbose text;
- it accepts both Python-style literals and JSON-style literals;
- it treats tuples and lists as equivalent for sequence outputs;
- for hashmaps, it allows key-value pairs within a bucket to appear in any order.

Before comparison, we also normalize the extracted output:

- remove extra whitespace, newlines, dollar signs, backslashes, and simple LaTeX spacing commands;
- normalize `\tfrac` and `\dfrac` to `\frac`;
- remove `\left`, `\right`, and degree markers;
- normalize decimal forms such as `.5` to `0.5`;
- normalize `null`/`None` and Boolean tokens;
- parse numeric answers as numbers, with a small tolerance for floating-point answers;
- recursively normalize nested sequence structures before equality checking.

An example is marked incorrect if the model produces a wrong final state, if no final answer can be extracted, or if the extracted answer cannot be parsed under these formatting relaxations. In preliminary experiments, we also tried Structured Output to allow more flexible answer formats while requiring JSON output, but this lowered performance. We therefore use flattened representations when possible, such as pre-order traversal for BST outputs, since a flat list is easier for models to produce reliably than a deeply nested structure. Future work could use an LLM judge to further reduce the effect of formatting constraints.

C.5 Reinforcement learning (RL) post-training

Training framework. We perform RL post-training with a rule-based outcome reward. The policy is initialized from an instruction-tuned model and trained with Group Relative Policy Optimization (GRPO) [23]. We use `verl` [24] as the RL training framework, which manages distributed policy updates, reference-policy log probabilities, advantage computation, and checkpointing. For generation during training, we use `vLLM` [14] as the rollout engine, which efficiently samples multiple responses from the current policy for each prompt.

For each prompt, the model samples multiple candidate responses. We extract the final boxed answer from each response and score it with the same deterministic verifier used for evaluation. A response receives reward 1 if the extracted answer matches the ground truth after normalization, and reward 0 otherwise. The scalar outcome reward is assigned to the final response token. We do not train a learned reward model; instead, the reward is computed directly from programmatic correctness.

Reward function. The reward function is a binary correctness signal aligned with the evaluation metric. Given a generated response, we first extract the final answer, prioritizing the final `\boxed{}` expression. We then apply the same formatting relaxations and normalization rules used in evaluation (Section C.4), such as stripping simple LaTeX wrappers, accepting JSON- or Python-style list literals, normalizing tuples to lists, and comparing parsed data-structure states rather than raw strings. The normalized prediction is compared against the programmatically generated ground truth. Correct outputs receive reward 1 and incorrect or unparsable outputs receive reward 0. This makes the RL signal deterministic and avoids using a learned reward model or an LLM judge during training.

Hyperparameters. Training uses a batch size of 128 prompts, with 5 rollouts per prompt for GRPO. The maximum prompt length is 2048 tokens. The maximum response length is 4096, 8192, and 16384 tokens for problem length ranges 5–10, 11–20, and 21–30, respectively. We train for 30 optimization steps with learning rate 5×10^{-6}. We use KL regularization against the reference policy with coefficient 0.001 and an entropy coefficient of 0.001. The PPO mini-batch size is 128, and the per-GPU micro-batch size is 4. Experiments were conducted on four NVIDIA H200 141GB GPUs.

C.6 Supervised finetuning (SFT) post-training

Data generation. We build SFT data from the same synthetic data-structure examples used for RL. For each generated problem, we construct a supervised prompt-response pair where the prompt contains the original data-structure question and the response contains a step-by-step solution trace followed by the final answer. For each data structure and operation type, we manually write SFT reasoning templates for insert and delete behavior. The data generator then fills these templates with the sampled values from each training example, producing a complete chain-of-thought reasoning trace that walks through the operation sequence and ends with the final serialized answer.

Example SFT training data. Below is an example array compound SFT instance with both insert and delete operations. After preprocessing, the text before `A:` is used as the user prompt, and the text after `A:` is used as the supervised assistant response.

```
[User prompt]
An array supports the following operations (using 0-based indexing):
1. (insert, index=i, value=v) inserts value v at position i,
   shifting elements at i and beyond to the right.
2. (delete, index=i) removes the element currently at position i.
Start with the provided initial array and apply each operation in order.
Initial array: [52, 74, 58, 7, 63, 79, 31]
Operations:
(insert, index=1, value=52)
(delete, index=3)
...
Q: What is the final array after applying all operations?
Your answer should be a list of integers, e.g. [1, 2, 3].

Approach the problem methodically. Ensure all conclusions are based on
precise calculations and logical deductions. Feel free to explore various
```

```
solution methods and cross-check results for consistency. Maintain dynamic
thinking and always verify each step of your reasoning. The last line of
your response should be of the following format: 'Therefore, the final
answer is: $\boxed{{ANSWER}}$.' (without quotes), where ANSWER is just the
final number or expression that solves the problem. Think carefully and
break down the problem step by step.

[Assistant response]
The original array is [52, 74, 58, 7, 63, 79, 31].
The first step is insert 52 at index 1.
Inserting 52 at index 1 shifts the elements from index 1 onwards to the right.
The updated array becomes [52, 52, 74, 58, 7, 63, 79, 31].
The next step is delete at index 3.
Removing the value 58 at index 3 updates the array.
The updated array becomes [52, 52, 74, 7, 63, 79, 31].
...
Therefore, the final answer is:
$\boxed{[52, 52, 73, 77, 7, 72, 63, 79, 68, 31]}$.
```

Training. We train for 300 optimization steps with learning rate 1×10^{-5}, cosine learning-rate scheduling, batch size of 32, weight decay 0.01, gradient clipping at 1.0, and 10% warmup.

C.7 The Berkeley Function Calling Leaderboard (BFCL) Benchmark

We use the Berkeley Function Calling Leaderboard (BFCL) [18] to evaluate whether a model produces the correct API call given a request. We focus on three Python function-calling categories: `simple_python`, `parallel`, and `parallel_multiple`. In BFCL, each function schema specifies a reusable skill primitive. The `simple_python` split contains atomic examples, where the user request can be solved by invoking a single function once. The `parallel` split contains compound examples that require multiple invocations of the same function schema, such as calling a weather API for several cities. The `parallel_multiple` split further increases compositionality by providing multiple function schemas and requiring the model to select and invoke several relevant functions. Thus, `simple_python` tests isolated skill execution, while `parallel` and `parallel_multiple` test whether the model can coordinate multiple skill invocations within a single user request.

Prompt. Each sample is converted into a chat-style prompt. The prompt gives the user request and the list of available Python functions, and asks the model to return the final function call or calls. If multiple calls are needed, the model is instructed to put one call per line. The gold answer is stored as a structured JSON payload containing the target calls and whether call order matters.

Data. We build separate atomic and compound training and test sets. The atomic training set contains 300 `simple_python` examples. The compound training set contains 150 `parallel` examples and 150 `parallel_multiple` examples. The test set contains 100 examples from each of the three categories, for 300 examples in total.

Verification. BFCL answers are verified with a rule-based exact-match checker. The checker parses model outputs from several accepted formats, including Python-style function calls, JSON objects or lists, `<answer>` tags, and the final `\boxed{}` answer format. Parsed calls are normalized before comparison: dictionary keys are sorted, numeric values are canonicalized, and unordered call sets are sorted when the BFCL example is not order-sensitive. A prediction receives reward 1 only if the normalized predicted function name, arguments, and call structure exactly match the normalized gold payload; otherwise it receives reward 0.

D Generalization of compositional reasoning

In Section 4, we primarily study *length generalization* to evaluate whether models can extend compositional reasoning beyond the training regime. To further probe the generalization induced by composed-task training, we consider two additional transfer settings: robustness under structural distribution shift, and transfer from basic skills to more complex tasks that require unseen skills.

D.1 Generalization under structural distribution shift

We test whether a model trained on one input distribution can generalize to others while the underlying skill remains unchanged. If the model has learned the skill rather than distribution-specific patterns, its performance should remain robust under such shifts.

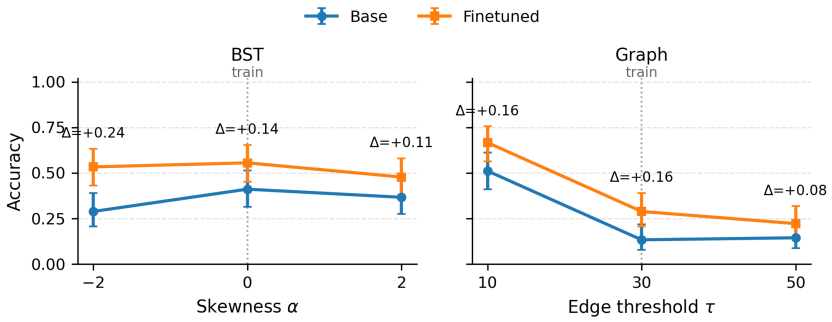


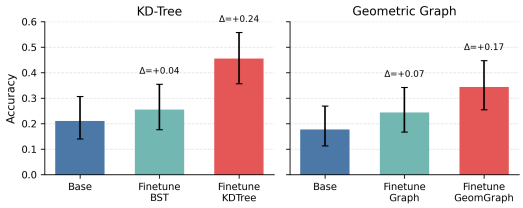
Figure 11: Distribution-shift generalization on BST and graph construction. Accuracy across shifts in α and τ. Dotted lines mark the training distribution; error bars show 95% confidence intervals.

Distribution-shift transfer. We study two tasks: binary search tree (BST) construction and graph construction. For BSTs, we introduce a parameter α to control tree skewness. When $\alpha > 0$, the generator favors values near the edges of the range, producing more unbalanced insertion orders and hence more skewed trees. When $\alpha < 0$, it favors values near the center, yielding more balanced trees. The case $\alpha = 0$ corresponds to the training distribution. For graph construction, the training distribution uses an edge threshold of 30: an edge is created when the absolute difference between two node values exceeds this threshold. At test time, lowering the threshold to 10 produces denser graphs, while increasing it to 50 produces sparser graphs.

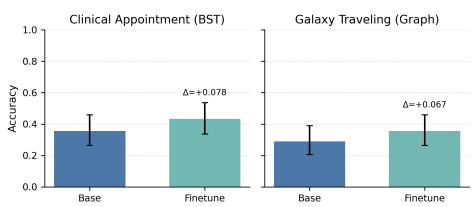
We find that the model finetuned with composed training remains robust under distribution shift, consistently outperforming the base model across all tested distributions in both BST and graph construction. However, the improvement is not uniform across distributions. For BSTs, the gain is largest on more balanced trees and smaller on more skewed trees, suggesting that skewed trees increase execution difficulty, plausibly by requiring a longer implicit stack. For graphs, the gain is also smaller in the sparse regime than in the denser or training-like regimes. Overall, composed training transfers robustly across structural distributions, but its benefit still depends on structural properties that affect execution difficulty.

D.2 Generalization to unseen skills

We study whether training on a set of basic skills enables a model to solve a more complex target task that requires skills unseen during training. We consider two settings.



(a) Transfer to structurally richer tasks requiring unseen skills.



(b) Transfer from formal tasks to natural-language variants.

Figure 12: Left: transfer to KD-Tree and Geometric Graph construction from basic 1D skills, compared with direct fine-tuning on the target task. Right: transfer from formal tasks to natural-language BST and graph variants. Error bars show 95% confidence intervals over test examples.

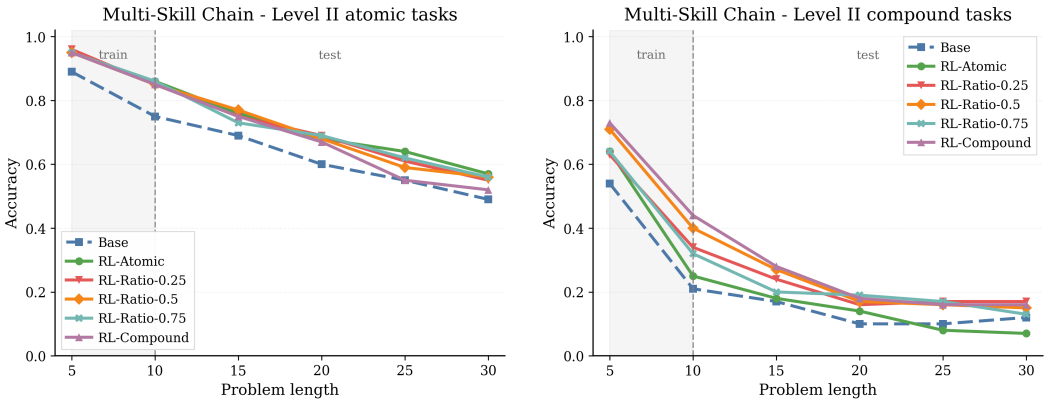
Higher-dimensional transfer. Following the spatial probe in DSR-Bench, we train on BST construction and evaluate on KD-Tree construction. Relative to BSTs, KD-Trees require an additional unseen skill: node-level splitting, where the model must partition a set of points along a specified dimension. For graph construction, we train on the 1D setting and evaluate on Geometric Graphs, a higher-dimensional variant in which edges are determined by Euclidean distance rather than absolute difference. In both cases, we use dimension $K = 2$.

Natural-language transfer. Following the realistic probe in DSR-Bench, we embed both tasks into more realistic natural-language settings. For BSTs, the task is framed as patients calling to book clinical appointments, where each node contains a patient name and appointment time. For graphs, the task is framed as spaceship travel across a galaxy through tunnels, where tunnels are created when two planets are sufficiently close in a hyperplane. These variants require the model to map from context-rich language to the underlying formal data-structure problem, and then execute the appropriate reasoning procedure.

We find that training on basic skills does not transfer strongly to harder target tasks that require unseen skills. On KD-Tree and Geometric Graph construction, the gains from transfer are small, while direct fine-tuning on the target task yields much larger improvements. Transfer to natural-language variants is also limited, with only modest gains over the base model. This suggests that compositional transfer remains difficult when generalization requires not only recombining known skills, but also acquiring new operations or reasoning primitives.

E Ablation on mixing atomic and compound training data

Our main controlled studies use atomic-only and compound-only training to examine the two endpoints in isolation, which allows us to identify the direction of transfer. To understand the role of mixed data in practical RL post-training, we conduct an ablation that varies the ratio of atomic tasks in the training data, keeping all other settings identical to the main study. As shown in Figure 13, the overall decomposed-to-composed asymmetry remains robust: including compound tasks improves compositional reasoning in cases where atomic tasks alone do not. However, the relationship between the mixing ratio and performance is not linear, so identifying the optimal mixture ratio is more nuanced; we leave a systematic investigation to future work.



(a) Different ratios of atomic training data.

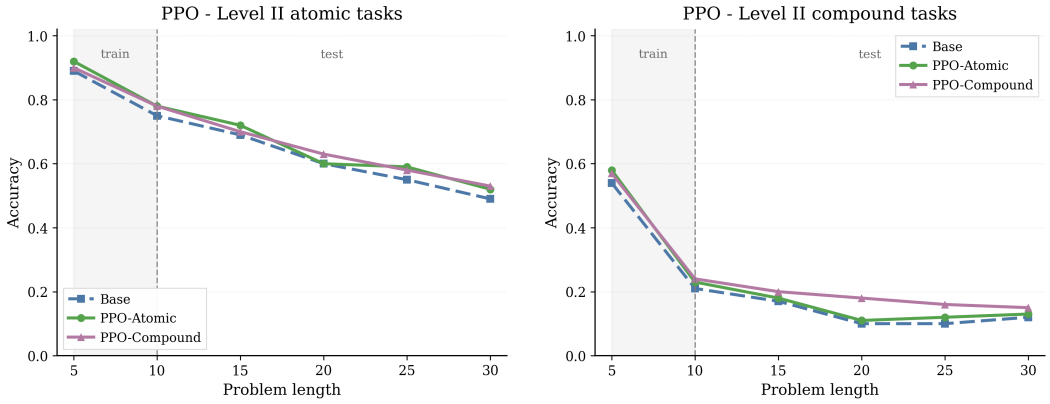
(b) Different ratios of atomic training data.

Figure 13: Ablation on the ratio of atomic tasks in the RL training data.

F Ablation on PPO

The primary goal of this paper is to isolate the effect of training-data composition rather than to compare RL algorithms. We use GRPO in the main study because it is widely adopted for LLM reasoning post-training and provides a consistent setting for comparing atomic and compound training. As a preliminary step, we conduct an ablation that replaces GRPO with PPO [22]. We use the default settings from GRPO and do not perform hyperparameter tuning for PPO; therefore,

the improvement over the base model is less significant. Nevertheless, as shown in Figure 14, the decomposed-to-composed asymmetry remains observable under a different RL algorithm.



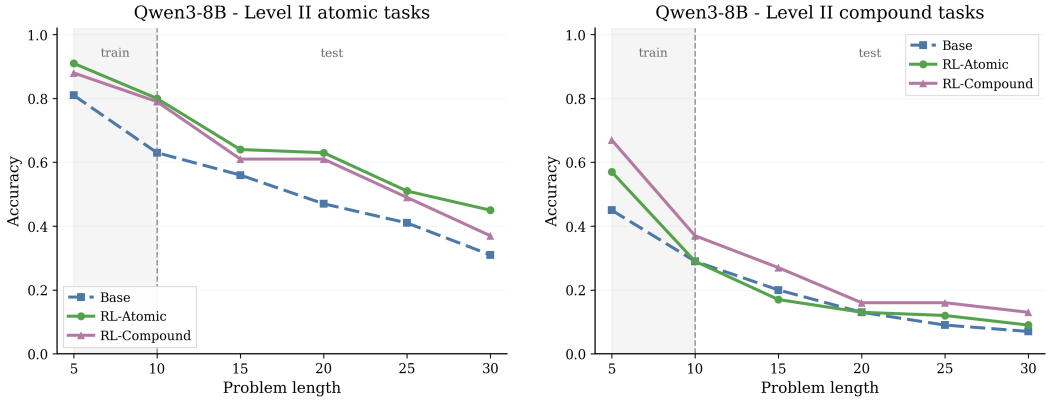
(a) PPO on Level II atomic tasks.

(b) PPO on Level II compound tasks.

Figure 14: Ablation replacing GRPO with PPO for RL post-training.

G Ablation: additional model scale

Our main experiments show that the decomposed-to-composed asymmetry is consistent across several models, including Qwen3-4B-Instruct-2507, OLMo-3-7B-Instruct, and Llama-3.1-8B-Instruct. However, because these models belong to different families, the effect of model scale cannot be fully isolated. We therefore conduct an additional experiment using Qwen3-8B with thinking disabled. Our primary model, Qwen3-4B-Instruct-2507, is an updated version of Qwen3-4B with non-thinking mode, whereas Qwen3-8B is from the original Qwen3 release. The corresponding July 2025 update includes the 4B, 30B, and 235B models but no 8B option. As the 30B and 235B models are mixture-of-experts models, Qwen3-8B, although not from the same release, is the closest feasible larger dense model. As shown in Figure 15, the decomposed-to-composed asymmetry remains robust at a larger scale within the Qwen3 family.



(a) Qwen3-8B on Level II atomic tasks.

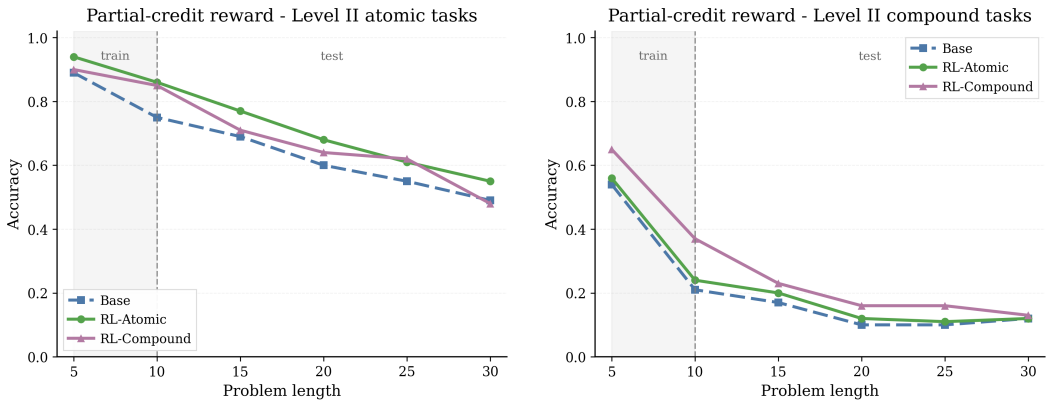
(b) Qwen3-8B on Level II compound tasks.

Figure 15: Results on Qwen3-8B (thinking disabled).

H Ablation: partial-credit reward

Our main study uses an exact, deterministic reward verifier to ensure a controlled experimental setting. To examine whether our findings hold under a noisier reward signal, we follow the alternative metric

used in DSR-Bench [10] and compute a partial-credit reward based on the Levenshtein distance between the model output and the ground truth, i.e., one minus the normalized edit distance. As discussed in DSR-Bench, binary scoring provides a clearer distinction and a fairer comparison than partial-credit rewards, which is why we adopt it in the main study. As shown in Figure 16, the overall trend remains robust under the partial-credit reward signal.



(a) Partial-credit reward on Level II atomic tasks. (b) Partial-credit reward on Level II compound tasks.

Figure 16: Results with a partial-credit reward based on normalized Levenshtein distance.

I Practical implications

Our observed decomposed-to-composed asymmetry has important practical implications for data design in RL post-training, especially as language models are increasingly trained to solve real-world problems that require advanced skills. Such problems usually require skills to be combined, not executed in isolation. In reasoning applications such as *proof writing*, compositional reasoning is required explicitly: success does not come from applying a single theorem or algebraic manipulation, but from choosing relevant lemmas, sequencing them into a valid argument, and maintaining dependencies between intermediate claims. Similar compositional reasoning also underlies real-world agentic applications. In *tool-calling agents*, the challenge is not only to learn individual API calls, such as retrieving weather, searching a database, or updating a calendar entry. The harder problem is to coordinate these calls into a valid workflow, where the output of one step determines what should happen next. Likewise, *coding agents* rarely succeed by editing one line in isolation: they must locate the relevant code, reason about dependencies across functions, apply a patch, and verify that the change is consistent with the intended behavior. Our findings suggest that although individual-skill training can be easier to supervise, it may not be sufficient for tasks that require composition. Training on composed tasks provides a more relevant signal for compositional generalization, while also transferring back to individual skills. Thus, for RL post-training on real-world deployment, data should prioritize verifiable multi-step trajectories rather than only isolated skill drills.

J Structured output and intermediate scoring

As described in Section C, we extract the final answer using a boxed output format, following Guo et al. [9]. To reduce errors caused by formatting issues, we additionally implement normalization and relaxation procedures, and evaluate outputs with instruction-tuned models. These design choices are based on preliminary experiments aimed at identifying the most suitable setup for evaluating compositional reasoning. Below, we discuss the rationale behind these choices.

One alternative is to use JSON-based Structured Output, a feature provided by `vllm` [14] that enforces outputs to follow a predefined JSON schema. This approach enables more flexible verification and allows correctness to be checked dynamically from successful executions, instead of relying on manually specified constraints such as always visiting neighbors in ascending order to guarantee a unique output. However, our preliminary experiments with Qwen3-4B-Instruct produces substantially fewer correct outputs when using Structured Output. We hypothesize that the stricter and more

complex JSON formatting requirements increase the generation difficulty. Therefore, we do not use Structured Output in our final setup, and instead evaluate serialized outputs (e.g., flattening a tree with preorder traversal) for correctness checking.

Another design choice is that we use only final-output scoring for both RL reward computation and evaluation. Since each step in the data structure computation is deterministic, intermediate states can in principle also be generated and verified. However, doing so requires additional formatting constraints to programmatically extract intermediate outputs. In our preliminary experiments, we found that adding such constraints consistently reduced overall performance. One possible reason is that models may rely on dynamic reasoning processes with self-correction (e.g., "Aha" moments), which are disrupted by enforcing structured intermediate outputs. As a result, we evaluate only the final output correctness for our experiments, providing flexibility to the models' reasoning process.

Finally, future work could explore LLM-based judging for answer extraction, removing the need for Structured Output or explicit intermediate-state formatting altogether. This may further reduce formatting-related errors. In addition, tracking intermediate reasoning trajectories and comparing them against ground-truth states—including cases with later error correction—could provide deeper insights into the compositional reasoning abilities of language models.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We elaborate on what related work has done regarding compositional reasoning on language models to the best of our knowledge, and summarize our contributions in abstract and introduction.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We include an explicit Limitations and future work section in Section 7.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We provide the detailed explanation, proof, including assumpstions for our theoretical claims in Section B.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide full experiment details and data generation procedures for reproducibility in Section C. We will make our code public upon acceptance.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: We will release a public repository containing our code upon acceptance. We provide full experiment details and data generation for reproducibility in Section C.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide full experiment details and data generation procedures in Section C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We repeat our experiments three times and report standard deviation shown as shaded region in Figure 3.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We declare the computing resources used in Section C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We confirm that we followed NeurIPS Code of Ethics when conduting research for this paper.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss practical implications of our paper in Section I.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper does not pose such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We cited all code, data, models used in this paper to the best of our knowledge.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: We do not introduce new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This research does not involve crowdsourcing nor human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: We did not involve crowdsourcing nor human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: We did not use LLM as an important or non-standard component for the core method development in this research.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.