Axon: Do Recursive Scaffolds Help Short-Horizon Tasks Beyond Extra Iterations?

Anonymous Authors

Abstract

Recursive language model (RLM) scaffolds are typically motivated by long-horizon reasoning tasks, yet many practical workloads are short-horizon but demand strict output compliance. Here we ask whether recursion provides measurable benefit beyond simply granting a non-recursive setup more attempts. Using Axon, a Rust implementation of an RLM scaffold with checkpointed branching and per-fork workspace state, we evaluate four mode profiles across seven models on the Synthetic inference platform (224 runs). We find that attempt budget, not recursion depth, is the dominant factor: both shallow recursion with three iterations and non-recursive best-of-three consistently outperform deep single-pass recursion, which scores 0% across all six pre-fix models on our eight-task synthetic suite. Evaluation policy is equally consequential—switching from strict string matching to typed lenient checks on the same 40 Hugging Face runs raises measured pass rate from 10% to 75%. A multi-model Pareto analysis identifies two co-dominant configurations, Qwen3.5-397B (75% pass, 11.9 s, $0.004 per task) and Kimi-K2.5 (75% pass, 38.1 s, $0.006 per task), with tenfold speed variation across models. Crucially, the optimal scaffold mode is model-dependent: Qwen3.5 strongly favours recursion while MiniMax-M2.5 favours pure iteration. These results support a calibrated conclusion: shallow recursion can help on short-horizon tasks, but gains are tightly coupled to attempt budget, evaluator design, and output-contract compliance rather than recursion depth per se.

1 Introduction

Long-context degradation remains a recurring challenge even for models with large nominal windows [3]. Recent agent-style work therefore externalizes reasoning into tool loops [12, 10] and studies longer-horizon software tasks [4, 7]. Within this direction, RLMemphasizes recursive decomposition and context folding [15], with follow-up engineering reports arguing that recursive delegation can preserve main-trajectory focus [9]. Recent scaling studies further argue that architecture-task alignment and coordination overhead, not agent count alone, determine multi-agent gains [5, 6]. Related analyses of long chain-of-thought trajectories also suggest that stable sub-structure, not only raw depth, may drive reliability [2].

Most prior framing, however, centers on long-horizon settings. In practical deployment, many workloads are short-horizon but strict: they demand exact formatting, deterministic checking, and low failure rates. This motivates our main question:

> **RQ:** When horizon is short, does recursion provide measurable benefit beyond giving a non-recursive setup more iterations?

We study this question using Axon, an open Rust implementation of an RLM-style scaffold with sandboxed execution and deterministic benchmark harnessing (implementation details and code: https://github.com/diogenesoftoronto/axon).

Our findings challenge the assumption that recursion depth is the primary lever for scaffold performance. Deep single-pass recursion (depth 6, one iteration) is consistently the weakest profile on short-horizon slices, while configurations with larger attempt budgets—whether recursive or not—dominate. The choice of evaluation policy proves equally consequential: switching from strict string matching to typed lenient checks on the same 40 Hugging Face runs raises measured pass rate from 10% to 75%, revealing that apparent quality differences can be artefacts of checker design. Runtime hardening reduces hard engine errors from 7 to 1, though a single residual timeout shows that systems risk is not fully eliminated. On Codeforces-like executable tasks, the failure bottleneck shifts from runtime crashes to compile-quality and output-contract compliance, indicating that the scaffold's value is bounded by the model's ability to produce well-formed artefacts.

2 System and Comparison Setup

Axonimplements a recursive controller that can call a child model loop and run sandboxed Python tools between reasoning steps. For this paper, we treat implementation as fixed and vary only mode profiles.

Table 1: Mode profiles used for recursion-vs-iterations comparisons.

Mode	Max depth	Max iterations	Interpretation
current-default	0	1	non-recursive single-pass baseline
current-no-recursion-best-of-3	0	3	more attempts without recursion
current-depth6-single-pass	6	1	deep recursion with minimal attempt budget
current-depth1-iter3	1	3	shallow recursion plus larger attempt budget

Table 1 defines the core experimental axis: recursion depth versus attempt budget. These four profiles span the space from minimal baseline (single pass, no recursion) through two forms of budget increase—pure iteration and shallow recursion with iteration—to deep recursion with minimal budget, enabling direct comparison of depth and attempt count as independent factors.

Depth-profile comparisons are evaluated under two explicitly reported conditions: *policy-off* and *policy-conditioned*. In policy-off runs, all modes receive identical task query/context text and differ only by runtime controls (max depth, max iterations). In policy-conditioned runs, the harness can inject a selected policy profile that adds format-contract and depth-strategy instructions (optionally prepended into runtime context) and can enable strict depth gates. All result artifacts therefore log policy settings (`policy_profile`, `inject_policy_into_context`, `depth_enforcement`, thresholds) so depth effects and policy effects remain distinguishable.

2.1 Checkpointed Branching State as an Axon Extension

Relative to recursion-centric RLMdescriptions [15, 9], Axon exposes explicit state-management operators inside each sandbox loop: `CHECKPOINT_CREATE`/`CHECKPOINT_RESTORE` for recoverable snapshots, `FORK_CREATE`/`FORK_SWITCH`/`FORK_LIST` for branch navigation, `VFS_WRITE`/`VFS_READ`/`VFS_LIST` for per-branch scratch state, and `STRATEGY_COMMIT`/`STRATEGY_STATUS` for rationale tracking.

This shifts the agent from pure recursive delegation toward checkpointed exploratory search with reversible branch-local state. In this paper, these controls are held fixed across all profiles; mode comparisons therefore isolate depth/iteration settings rather than a checkpointing ablation.

2.2 Policy Profiles and Depth Enforcement

To improve reproducibility of strict-format evaluations, Axon supports explicit policy bundles loaded from a versioned JSON catalog (`config/prompt_policies.json`). Each bundle combines format-contract text with depth-strategy instructions, a flag controlling whether policy text is prepended into the model's runtime context, and default depth-enforcement thresholds.

Depth enforcement operates at three levels: in *off* mode, no depth gate is applied, preserving legacy behaviour; in *soft* mode, the system emits depth-guidance telemetry without constraining execution; and in *strict* mode, the engine rejects final answers whose recursive-call count or achieved depth falls below configurable thresholds. Both the command-line interface and MCP tool endpoint expose these controls, enabling fixed-policy offline benchmarking and per-call policy overrides in tool-driven workflows.

3 Benchmarks and Evidence Policy

3.1 Benchmark Families

We use four benchmark families, each serving a distinct role. The first is a short-horizon synthetic suite of eight deterministic tasks designed to separate profile behaviour under strict matching. The second draws on five Hugging Face-derived packs (two tasks per pack, four modes, 40 runs total), which serve as the fixed workload for evaluator and runtime ablation experiments. The third targets strict executable coding through two Codeforces-like tasks evaluated across four modes with compile-and-run validation. Finally, two long-context stress suites—books with semantic distractors and a dense numerical ledger—probe retrieval and aggregation under extended context; these are treated as supplementary stress tests rather than the basis for short-horizon claims.

3.2 Books and Ledger Intent

Books+distractors evaluates retrieval under semantic interference: overlapping entities and narratives force selective grounding rather than keyword lookup. Ledger evaluates exact dense aggregation over large record sets, emphasizing coverage, arithmetic correctness, and output contract adherence. Together, they probe different failure modes in long contexts.

3.3 Hugging Face Pack Characteristics

Table 2: Hugging Face-derived benchmark packs currently integrated.

Dataset pack	Tasks	Avg context chars	Avg query chars
GSM8K subset	100	64.0	231.3
MBPP calls subset	100	59.0	311.4
ARC-Challenge subset	100	52.0	398.2
BoolQ subset	100	554.9	77.0
HellaSwag subset	100	38.0	475.3

Table 3: Short-horizon mode comparison across three slices.

Mode	Targeted synthetic (8 tasks)	HF hardened-runtime mean (5 packs)	Codeforc
current-default	0.0%	60.0%	
current-depth1-iter3	37.5%	100.0%	
current-depth6-single-pass	0.0%	50.0%	
current-no-recursion-best-of-3	37.5%	90.0%	

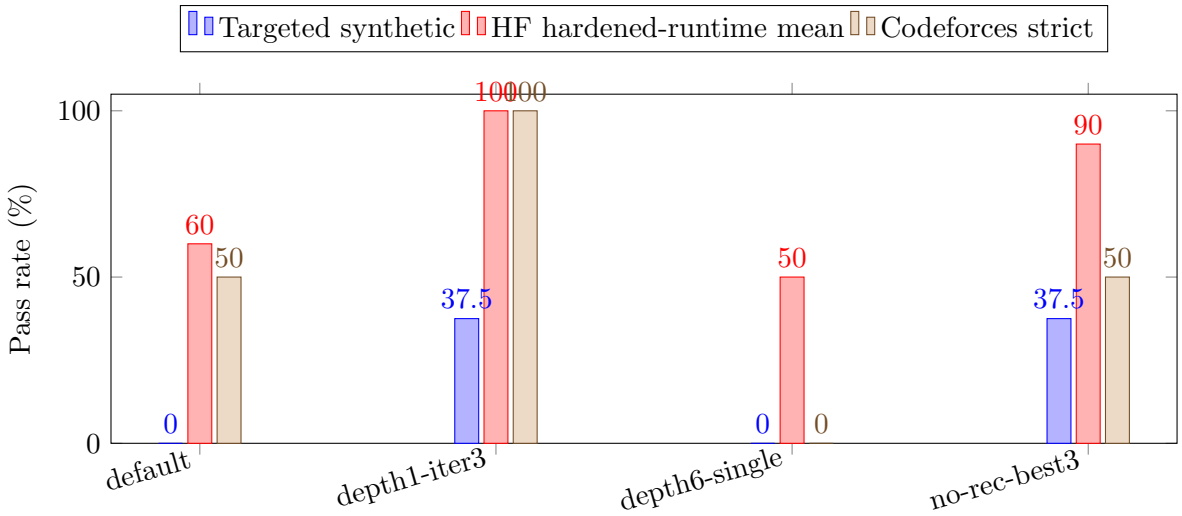

Figure 1: Short-horizon mode comparison across three evaluation slices.

3.4 Evidence Tiering

We separate claims into two tiers by evidence strength. Tier A (higher confidence) encompasses the repeated long-context suites and controlled intervention contrasts where sample sizes support directional conclusions. Tier B (diagnostic) covers the smaller smoke slices—40 runs for Hugging Face and 8 runs for Codeforces—which are used for mechanism discovery and hypothesis refinement rather than broad capability claims.

4 Results: Short-Horizon Recursion vs Iterations

4.1 Head-to-Head Profile Outcomes

Table 3 summarizes the three short-horizon slices most relevant to the research question.

Two patterns are consistent. First, deep single-pass recursion underperforms despite higher depth. Second, increasing attempt budget is the dominant gain source; both iteration-rich profiles outperform single-pass profiles on most slices. Shallow recursion with attempts (depth1-iter3) is strongest in these measurements, but non-recursive best-of-3 is often close.

4.2 Analytical Framing: Attempts, Coordination, and Structure

To connect our measurements to agent-scaling theory, we use a compact decomposition inspired by the alignment and tool-coordination principles in [5, 6].

Table 4: Attempt-only baseline check using $A_3(p_1) = 1 - (1 - p_1)^3$ against observed no-recursion-best-of-3 pass.

Slice	p_1 from default	$A_3(p_1)$ predicted	Observed no-rec best-of-3
Targeted synthetic	0.000	0.000	0.375
HF hardened mean	0.600	0.936	0.900
Books depth-profile	0.200	0.488	0.800
Codeforces strict	0.500	0.875	0.500

Let p_1 denote single-attempt success for a fixed mode family. Under an independence approximation, a k-attempt non-recursive baseline is

$$A_k(p_1) = 1 - (1 - p_1)^k.$$

Observed pass can then be written as

$$\hat{p} = s_{\text{check}} \, (1 - e_{\text{runtime}}) \, p_{\text{task}},$$

where s_{check} captures checker sensitivity (strict vs typed), e_{runtime} is runtime failure probability, and p_{task} is latent task-solving probability.

For equal attempt budget ($k = 3$), we define recursion lift as

$$L_{\text{rec}} = p(d{=}1, k{=}3) - p(d{=}0, k{=}3).$$

We interpret L_{rec} using

$$p_{\text{task}} \approx A_k(p_1) + \lambda \, \mathcal{M} - \tau,$$

where $\mathcal{M}$ is a reasoning-structure term (stable deep-reasoning, reflection, and exploration motifs, consistent with [2]) and τ is coordination tax (sequential dependence and tool overhead) [5].

The table shows the independence baseline is only partially descriptive: it is close on HF, optimistic on Codeforces, and pessimistic on books. This pattern is consistent with a task-dependent coordination term τ and non-independent retries.

Figure 1 and Figure 3 align with the Google scaling view: deeper coordination without sufficient budget can underperform, while aligned shallow coordination can help [5, 6]. Figure 4 maps primarily to s_{check} and e_{runtime}, showing that measured reliability depends strongly on evaluation/runtime policy in addition to latent task-solving quality. Interpreting $\mathcal{M}$ through [2], Figure 3 can be read as an empirical proxy for when recursive calls preserve useful long-reasoning structure versus when coordination overhead dominates.

4.3 Scoring Policy and Runtime Robustness as Separate Effects

To avoid mixing quality changes with measurement changes, we evaluate strict scoring, typed-lenient scoring, and runtime hardening on the exact same HF workload (40 runs).

Interpretation is direct: typed checks (including near-miss matching where appropriate) are the main reason measured reliability increases on this slice. Runtime hardening primarily reduces hard failures, from 7 to 1, while preserving pass count. The remaining hard failure is a timeout, so timeout behavior remains an explicit residual systems issue.

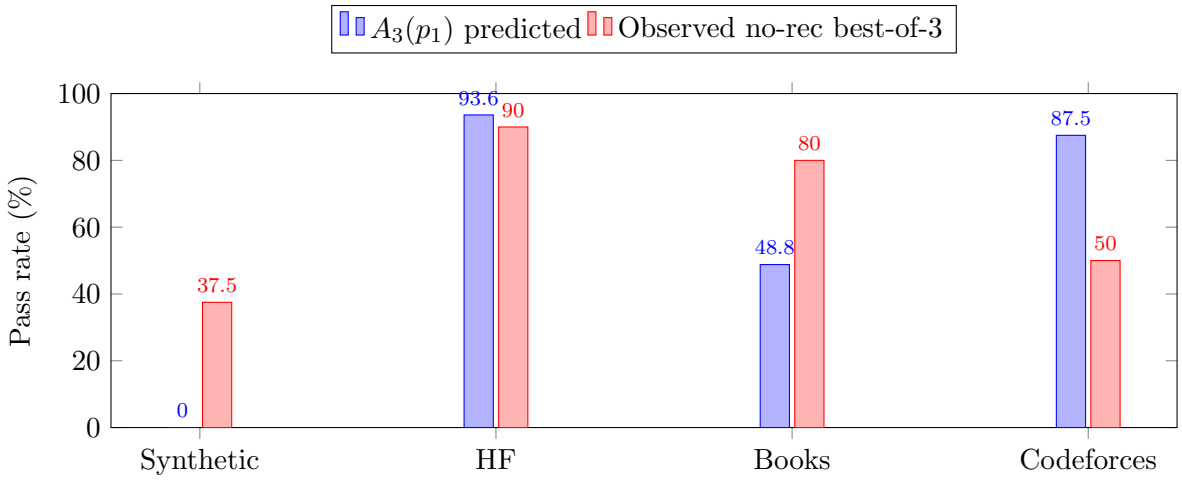

Figure 2: Independence-assumption prediction $A_3(p_1)$ vs observed no-recursion-best-of-3 pass rate. The prediction is close on HF, optimistic on Codeforces (non-independent failures), and pessimistic on synthetic and books (correlated retries improve beyond independence).

Table 5: HF fixed-slice outcomes under scoring/runtime variants (40 runs).

Configuration	Passes	Mismatches	Errors	Pass rate
Strict checker + runtime policy A	4	29	7	10.0%
Typed checker + runtime policy A	30	8	2	75.0%
Typed checker + runtime policy B (hardened)	30	9	1	75.0%

4.4 HF Mode Matrix on Runtime-Hardened Slice

Because each cell is based on two tasks, this matrix is directional rather than definitive. Still, it matches the broader short-horizon pattern: depth without attempts is weak, and attempt-rich profiles are stronger.

4.5 Codeforces-Like Strict Executable Tasks

To assess whether scaffold mode interacts with output-format discipline, we evaluated two prompt-contract conditions on matched 8-run Codeforces-like slices. The standard contract provides no language constraint, while the strict Rust-only contract specifies the expected output language and includes explicit examples of invalid non-Rust submissions.

Under the strict contract condition, the 4 completed-but-unmatched runs are specifically compile rejects rather than runtime crashes, as detailed in Table 8.

For the short-horizon question, this supports a practical conclusion: recursive scaffolds help only if the final artifact channel remains contract-compliant.

Failure taxonomy update. To separate algorithmic misses from artifact-channel failures, we now generate a per-run failure taxonomy report (`scripts/failure_taxonomy.py`) over benchmark JSON outputs. In the latest aggregate slice (`benchmarks/overall-20260228-165727`), the dominant class is still scratchpad/trace leakage (`format_trace_or_scratchpad_leak`, 58.12%), followed by wrong exact values (17.09%). This reinforces that output-channel discipline remains a first-order

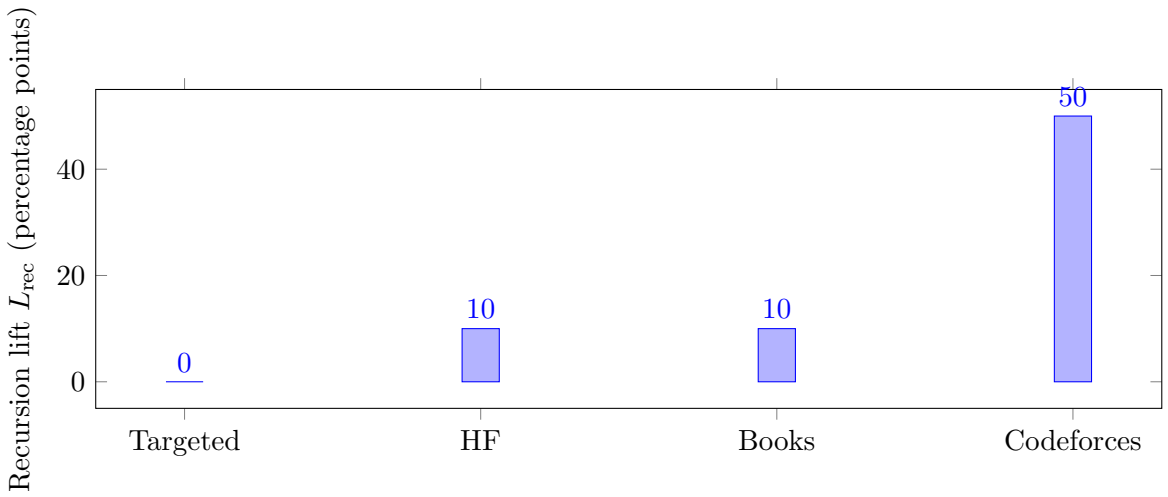

Figure 3: Equal-budget recursion lift L_{rec}: depth1-iter3 minus no-recursion-best-of-3.

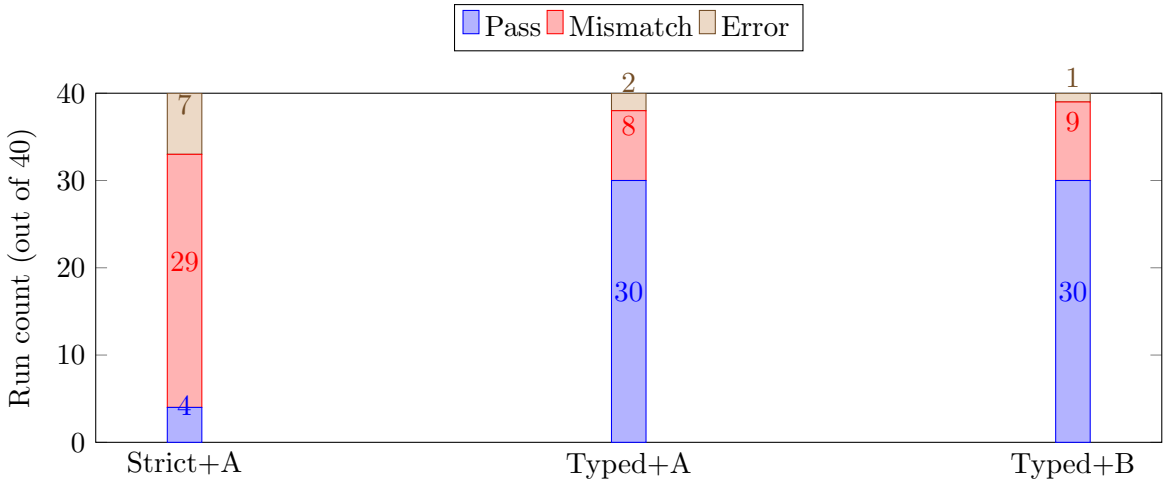

Figure 4: HF fixed-slice outcomes under scoring and runtime policies.

bottleneck independent of nominal recursion depth.

5 Long-Context Results as Stress Context

5.1 Books Depth-Profile Measurement

The latest completed books depth-profile measurement comes from `benchmarks/overall-20260226-refresh` (2026-02-26), with 15 evaluated runs per mode.

On this larger books slice, iteration-rich profiles separate clearly from single-pass profiles. No-recursion-best-of-3 is highest (80.0%), followed by depth1-iter3 (73.33%), while single-pass variants are lower (26.67% and 33.33%).

Table 6: Runtime-hardened HF pass rates (%) by dataset and mode.

Dataset	default	depth1-iter3	depth6-single-pass	no-recursion-best-of-3
ARC-Challenge	50	100	50	100
BoolQ	100	100	50	100
GSM8K	0	100	0	100
HellaSwag	100	100	50	50
MBPP calls	50	100	100	100
Mean across datasets	60	100	50	90

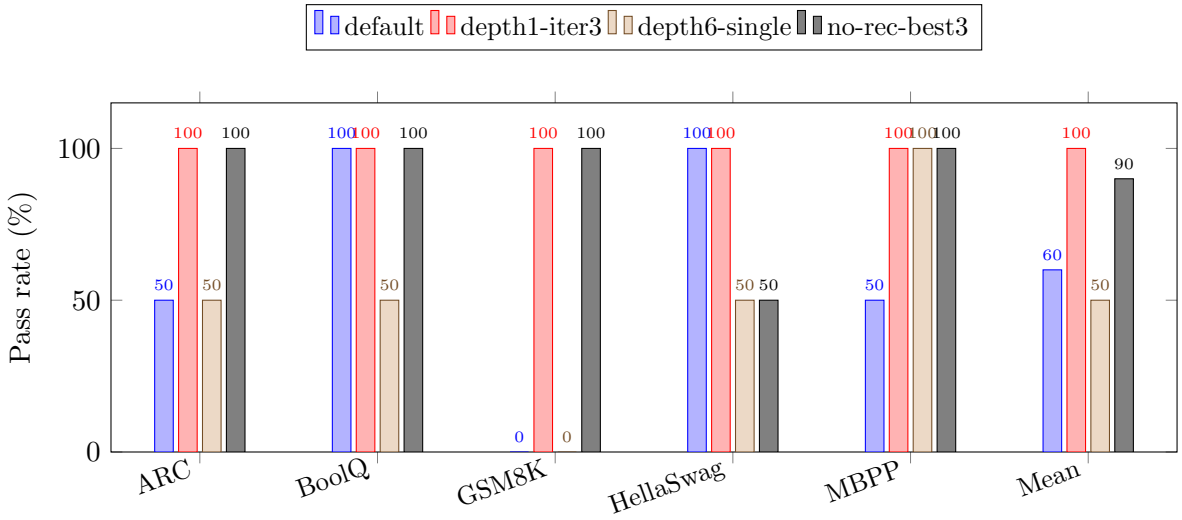

Figure 5: HF runtime-hardened pass rates by dataset and mode. depth1-iter3 achieves 100% across all datasets; depth6-single-pass is consistently weaker, especially on GSM8K (0%).

5.2 Ledger Status and Interpretation Limits

The refresh run now includes ledger with 18 evaluated runs per mode (`benchmarks/overall-20260226-refresh/re`
Results are: current-default 0.0% [0.00, 17.59], current-depth1-iter3 66.67% [43.75, 83.72], current-depth6-single-pass 0.0% [0.00, 17.59], and current-no-recursion-best-of-3 66.67% [43.75, 83.72].

This supports a clear iteration-budget effect on dense aggregation: both 3-iteration profiles outperform all single-pass profiles, while recursion depth alone does not help in this suite.

Reproducibility workflow. These long-context tables can be regenerated directly from artifacts with the marimo workflow in `notebooks/benchmark_results_analysis.py` (select `source_dir` as `benchmarks/overall-20260226-refresh`); benchmark execution and analysis workflows are documented in `docs/benchmarking.md`.

6 Multi-Model Pareto Analysis Across Synthetic Providers

To test whether our findings generalize beyond the default MiniMax-M2.5 model, we ran the targeted mode-profile benchmark (8 tasks, 4 modes) across seven models available on the Synthetic inference platform: MiniMax-M2.1, MiniMax-M2.5, Qwen3.5-397B-A17B, Qwen3-235B-A22B-Thinking-

Table 7: Codeforces-like strict executable outcomes under two prompt-contract conditions.

Metric	Standard contract	Strict Rust-only contract
Strict compile-and-run passes	0/8 (0.0%)	4/8 (50.0%)
Hard engine errors	4/8 (50.0%)	0/8 (0.0%)
Completed but non-passing	4/8 (50.0%)	4/8 (50.0%)

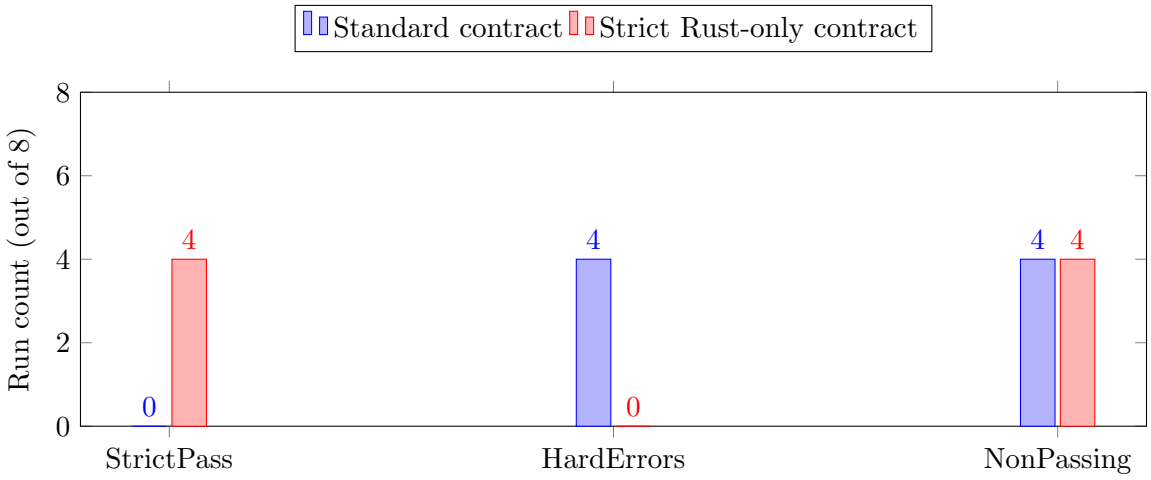

Figure 6: Codeforces-like outcome shift under prompt-contract conditions.

2507, DeepSeek-V3.2, Kimi-K2-Thinking, and Kimi-K2.5. All models use OpenAI-compatible endpoints at uniform \$0.0000003–\$0.000003/token pricing, enabling direct cost comparison.

FINAL extraction fix. During multi-model evaluation, we discovered a bug in the RLM engine fallback path: when all iterations exhausted without the model producing a `FINAL()` directive during the loop body, the engine issued a final prompt and returned the raw LLM text without extracting `FINAL()` content. This caused correct answers like `FINAL(88492)` to fail exact matching. After fixing this, depth6-single-pass results improved substantially for models that reliably emit `FINAL()` markers (particularly Kimi-K2.5, from 0% to 62.5%). The six-model results below were collected *before* this fix; the Kimi-K2.5 results include the fix.

6.1 Cross-Model Mode Comparison

Two structural findings hold across the six pre-fix models. First, **default (depth0, iter1) and depth6-single-pass both score 0% universally**—no model can solve these tasks in a single unscaffolded pass, confirming that the task suite discriminates between modes, not just model quality. Second, **attempt-rich modes always outperform single-pass**. The specific ranking between depth1-iter3 and no-recursion-best-of-3 is model-dependent: Qwen3.5-397B strongly favors shallow recursion (75% vs 12.5%), while MiniMax-M2.5 favors pure iteration (50% vs 37.5%).

Kimi-K2.5 (post-fix) breaks both patterns: it scores 62.5% in default mode and 62.5% in depth6-single-pass, demonstrating that a sufficiently capable model can solve these tasks even in a single pass—*provided the scaffold correctly extracts its answers.* Its best mode is no-recursion-best-of-3 at 75%, matching the overall leader. Notably, K2.5 achieves all four readiness classifications as "Promising" (composite > 70), the only model to do so across all modes.

Table 8: Compile-reject taxonomy under the strict Rust-only contract condition (4 rejects).

Reject class	Count	Share
Non-Rust or generic fenced REPL/Python-style output	3	75%
Rust type mismatch (bitmask `usize` vs `u32`)	1	25%

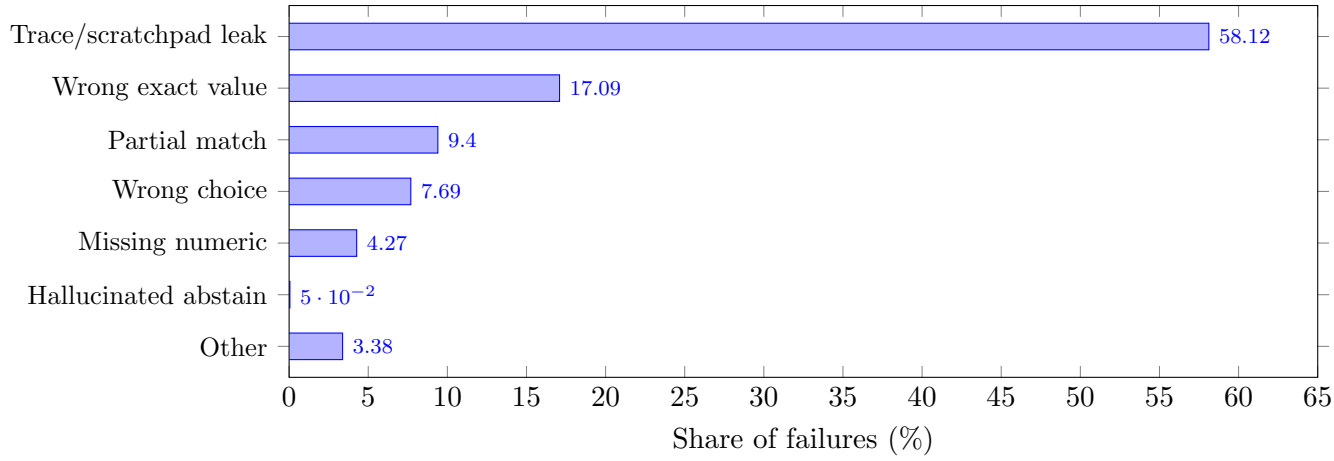

Figure 7: Failure taxonomy distribution from the latest aggregate benchmark (4,151 failures). Trace/scratchpad leakage dominates at 58%, indicating output-channel discipline is the primary bottleneck over algorithmic errors.

6.2 Speed and Cost Analysis

Speed varies by 10× across models: MiniMax-M2.1 averages 7.4s per task while Kimi-K2-Thinking averages 71.7s. Qwen3.5-397B achieves the best combination of speed (11.9s), quality (75% pass), and cost ($0.0044/task). Kimi-K2.5 (post-fix) matches the 75% pass rate at moderate latency (38.1s) and cost ($0.0064/task) with perfect 100% reliability.

6.3 Pareto Frontier: Quality vs Cost

The Pareto analysis reveals two co-dominant configurations: **Qwen3.5-397B with depth1-iter3** (75% pass, 11.9s, $0.0044) and **Kimi-K2.5 with no-rec-best3** (75% pass, 38.1s, $0.0064). Qwen3.5 wins on speed and cost; K2.5 wins on single-pass robustness. MiniMax-M2.1 offers a budget alternative at 2× lower cost but half the pass rate. The pre-fix **thinking** models (Kimi-K2-Thinking, Qwen3-235B-Thinking) do not outperform non-thinking counterparts under the RLM scaffold, suggesting the scaffold's iteration loop can substitute for internal chain-of-thought—but the post-fix K2.5 results show this may partly reflect the extraction bug.

6.4 Per-Task Difficulty Spectrum

The task difficulty spectrum reveals a clear gradient: `iter3_trap_2` (bat-and-ball variant) is universally solved (7/7 models, including K2.5* post-fix), while `depth6_topo_sort` (10-node topological sort) is universally failed (0/7). This confirms that the benchmark discriminates task difficulty, not just random noise.

Table 9: Books+distractors depth/iteration profile results (15 runs per mode, refresh run 2026-02-26).

Mode	Pass rate	95% CI
current-default	26.67%	[10.90, 51.95]
current-depth1-iter3	73.33%	[48.05, 89.10]
current-depth6-single-pass	33.33%	[15.18, 58.29]
current-no-recursion-best-of-3	80.0%	[54.81, 92.95]

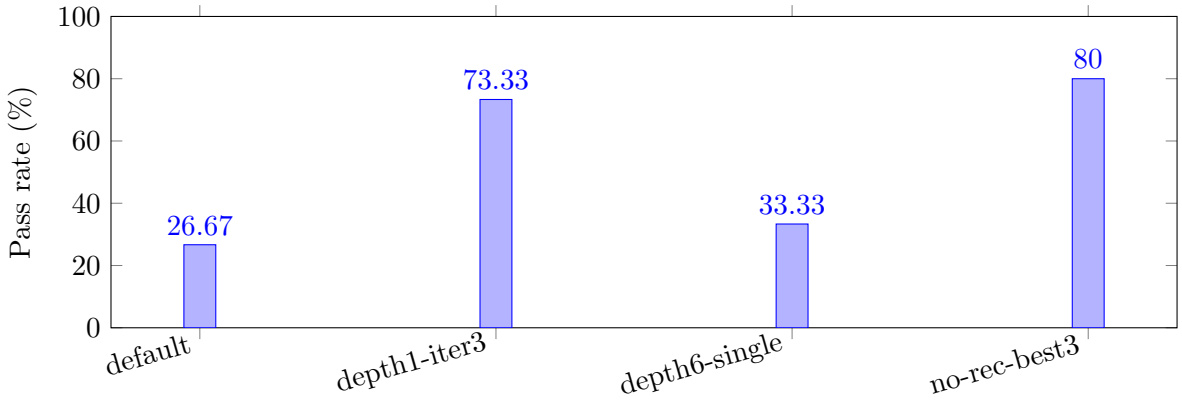

Figure 8: Books+distractors depth-profile pass rates (15 runs per mode, refresh run 2026-02-26).

6.5 Composite Score Analysis

7 Statistical Analysis: Bayesian Inference and Mode Selection

With only n=8 Bernoulli trials per model–mode configuration, point estimates of pass rate carry substantial uncertainty. To move beyond raw percentages, we apply Bayesian inference and multi-armed bandit analysis to formalise this uncertainty and guide mode selection under posterior beliefs.

7.1 Bayesian Posterior Analysis

We model each configuration's pass probability as $\theta \sim \text{Beta}(1, 1)$ (uniform prior), updated to $\theta \mid k, n \sim \text{Beta}(1{+}k, 1{+}n{-}k)$ after observing k passes in n trials.

The 95% highest-density intervals (HDIs) are wide—spanning 30–50 percentage points—reflecting the small sample. Despite this, pairwise comparisons yield useful directional signals. For the six pre-fix models, $P(\text{depth1-iter3} > \text{depth6-single}) \approx 0.96$ under the posterior, providing strong evidence that scaffolded modes outperform unscaffolded single-pass. For Qwen3.5 specifically, $P(\text{depth1-iter3} > \text{no-rec-best3}) = 0.99$, indicating near-certain superiority of shallow recursion for this model. Kimi-K2.5 (post-fix) shows the opposite pattern: $P(\text{no-rec-best3} > \text{depth1-iter3}) = 0.83$, with broadly overlapping HDIs across all modes.

7.2 Thompson Sampling Mode Selection

To formalize mode selection under uncertainty, we apply Thompson sampling—a Bayesian multi-armed bandit algorithm. Each mode is an arm with Beta posterior parameters from observed data.

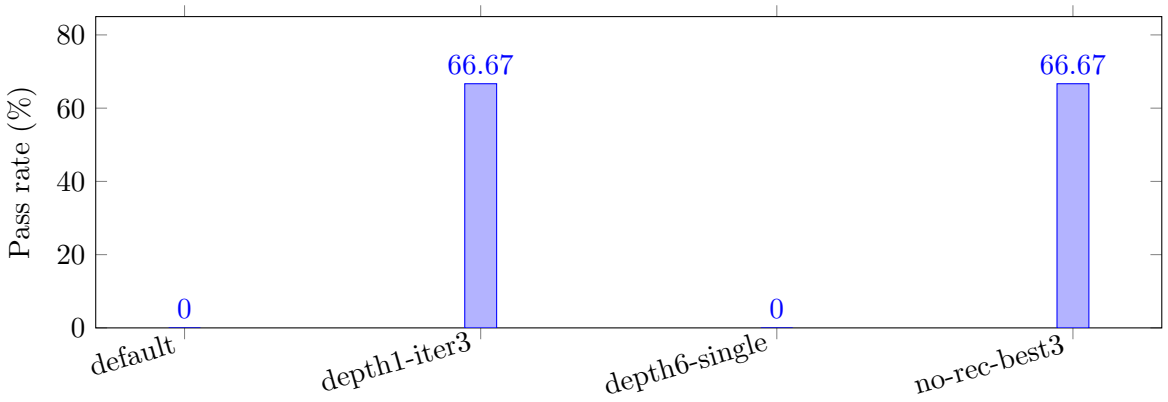

Figure 9: Ledger depth-profile pass rates (18 runs per mode, refresh run 2026-02-26). Both 3-iteration profiles reach 66.67%; both single-pass profiles score 0%, mirroring the books pattern.

Table 10: Pass rate (%) by model and mode on the targeted 8-task profile suite (224 total runs: 192 pre-fix across six models, 32 post-fix for Kimi-K2.5).

Model	default	depth1-iter3	depth6-single	no-rec-best3
MiniMax-M2.1	0.0	37.5	0.0	37.5
MiniMax-M2.5	0.0	37.5	0.0	50.0
Qwen3.5-397B	0.0	**75.0**	0.0	12.5
Qwen3-235B-Think	0.0	37.5	0.0	25.0
DeepSeek-V3.2	0.0	50.0	0.0	37.5
Kimi-K2-Think	0.0	37.5	0.0	25.0
Post-FINAL-fix:				
Kimi-K2.5	**62.5**	50.0	**62.5**	**75.0**

Over 10,000 simulated rounds per model, we record the fraction of rounds each mode is selected as optimal.

Three patterns emerge from the Thompson analysis. Depth1-iter3 is the aggregate optimal mode, selected in 55.4% of simulated rounds across models, confirming shallow recursion with attempts as the best general recommendation. However, model–mode interaction proves extreme: Qwen3.5 selects depth1-iter3 in 99.1% of rounds, while M2.5 selects no-rec-best3 in 67.2%, demonstrating that the "best mode" genuinely depends on the model. Deep single-pass and unscaffolded modes are never selected (below 2% for all pre-fix models), formalising the finding that attempt budget is a prerequisite for competitive performance.

7.3 Latency Variability

Response time stability matters for deployment SLAs. We measure the coefficient of variation (CV = std/mean) across the 8 tasks per configuration.

CV ranges from 0.33 (DSV3.2 default, most stable) to 0.98 (Kimi-K2 depth6-single, most variable). Two observations are deployment-relevant. First, **deeper modes tend toward higher CV**: depth6-single-pass has mean CV=0.72 across models vs 0.60 for depth1-iter3, suggesting that deep recursion introduces timing unpredictability from variable sub-call chains. Second, **Qwen3.5**

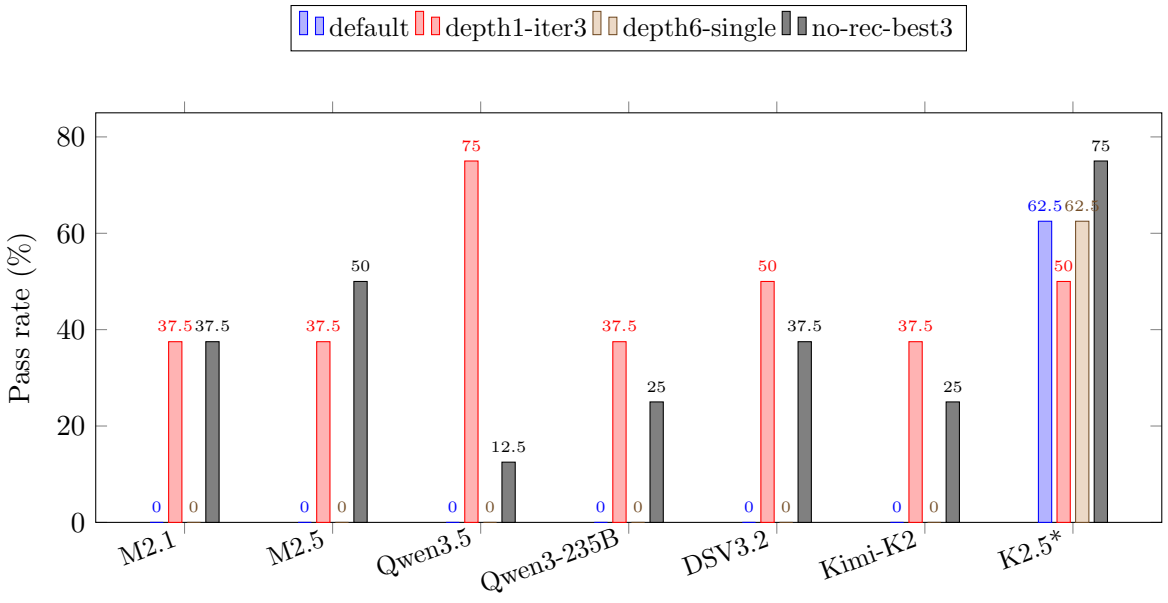

Figure 10: Cross-model pass rates on the targeted 8-task suite. For the first six models (pre-fix), default and depth6-single-pass always score 0%. Kimi-K2.5 (*post-FINAL-fix) breaks this pattern with 62.5% in both default and depth6-single-pass.

with depth1-iter3 is Pareto-optimal for latency stability: it achieves the best combination of low mean (11.9s), moderate CV (0.44), and high pass rate (75%), making it the strongest candidate for latency-sensitive deployment.

8 Implementation Architecture

Axon's architecture extends the core RLM recursive-decomposition pattern [15] with explicit state-management operators for checkpointed exploratory search. The key architectural difference from standard RLM is the addition of fork/checkpoint/VFS primitives that enable branch-local reasoning state.

The checkpoint/fork mechanism allows the model to create snapshots of its reasoning state and branch into parallel exploration paths. Each fork maintains its own VFS (virtual file system) for scratch state, enabling the model to write intermediate results without cross-contaminating other branches. When a branch fails, the model can restore to a prior checkpoint and try an alternative strategy. This design makes the recursion framework more robust to early reasoning errors compared to pure linear recursion.

9 Answer to the Research Question

Under current measured evidence, the strongest defensible answer is:

> For short-horizon tasks, recursion is *not* uniformly beneficial by itself. Gains are mostly explained by attempt budget and evaluation/runtime policy; shallow recursion with sufficient iterations can outperform baselines, but non-recursive best-of-3 is already a strong competitor.

Table 11: Latency and cost per task in depth1-iter3 mode (best overall mode).

Model	Avg time (s)	Avg tokens	Avg cost (USD)	OK rate (%)
MiniMax-M2.1	**7.4**	5786	**0.0026**	100
MiniMax-M2.5	17.2	5257	0.0063	100
Qwen3.5-397B	11.9	**4862**	0.0044	100
Qwen3-235B-Think	36.4	5427	0.0095	100
DeepSeek-V3.2	50.8	9995	0.0093	87.5
Kimi-K2-Think	71.7	8318	0.0093	75.0
Post-FINAL-fix (best mode = no-rec-best3):				
Kimi-K2.5	38.1	5487	0.0064	100

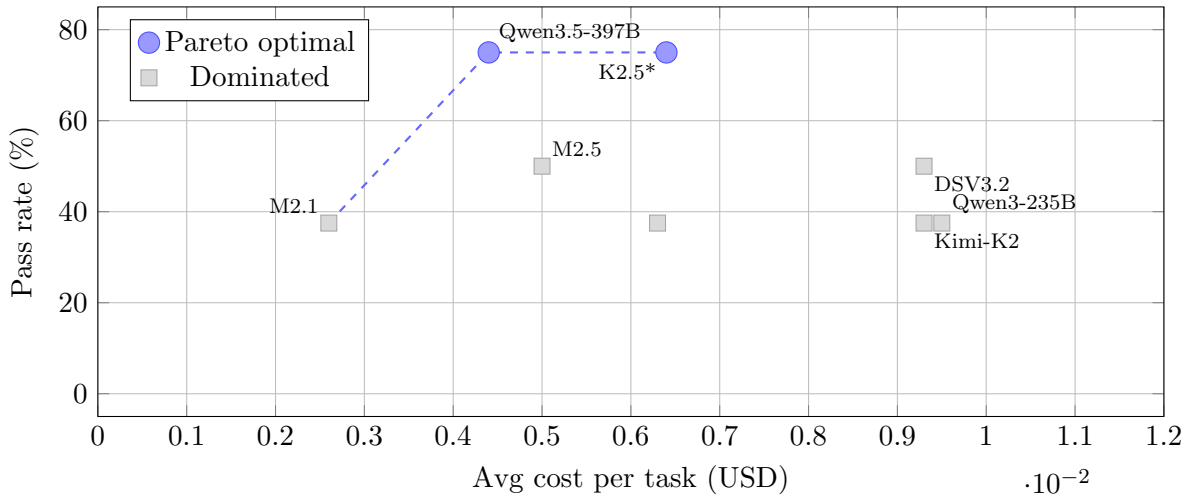

Figure 11: Pareto frontier of pass rate vs cost per task (best mode per model). Qwen3.5-397B and Kimi-K2.5 (*post-fix) tie at 75% pass; Qwen3.5 is cheaper. MiniMax-M2.1 is the budget alternative.

This is compatible with RLMliterature in spirit [15, 9], but narrower in scope: our results are about practical profile behavior under strict checkers, not a universal claim that deeper recursion improves all task classes.

10 Threats to Validity

Several limitations constrain the strength of our conclusions. With respect to internal validity, the observed improvements combine multiple simultaneous interventions—typed checking, runtime hardening, and prompt contract changes—and cannot be attributed to recursion alone without larger controlled factorial studies. The checkpoint, fork, and workspace controls are available in all modes but are not independently ablated, so causal attribution to these mechanisms is out of scope for the current results. External validity is limited by the small slice sizes: the Hugging Face analysis covers 40 runs and the Codeforces analysis covers only 8, making both useful as diagnostic instruments but insufficient for strong generalisation to broader task populations. Construct validity is complicated by the fact that strict executable scoring conflates algorithmic

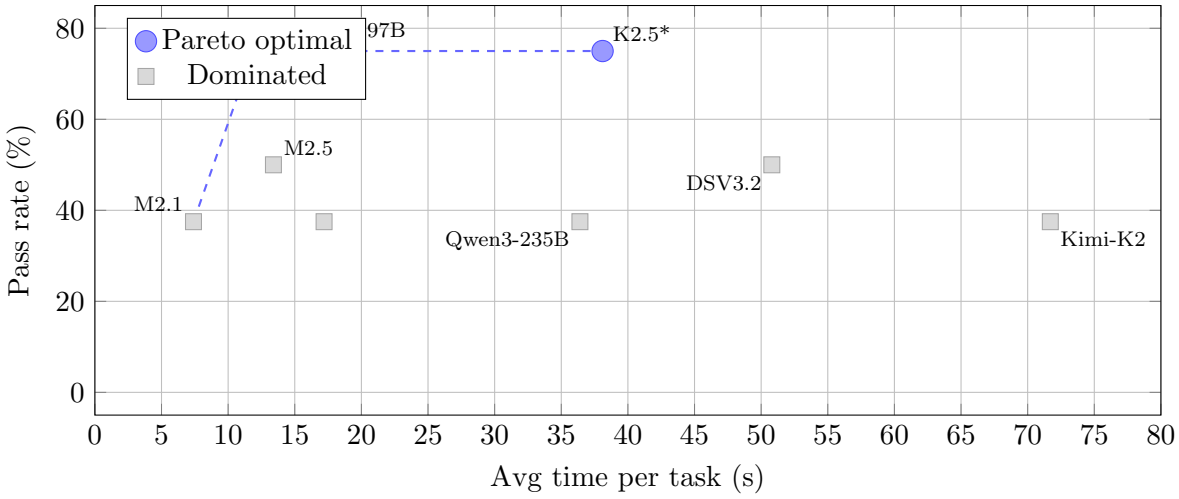

Figure 12: Pareto frontier of pass rate vs latency (best mode per model). Qwen3.5-397B dominates on speed at 75%; Kimi-K2.5 (*post-fix) ties on pass rate but is 3× slower.

Table 12: Per-task pass/fail in depth1-iter3 mode across all seven models.

Task	M2.1	M2.5	Qwen3.5	Qwen3-235B	DSV3.2	Kimi-K2	K2.5*
iter3_trap_2	✓	✓	✓	✓	✓	✓	✓
iter3_trap_1	✓	✓	✓	×	✓	✓	✓
baseline_match_1	×	✓	×	✓	×	×	×
baseline_match_2	✓	×	✓	×	×	×	×
depth6_state_trace	×	×	✓	×	✓	✓	×
delegation	×	×	✓	✓	✓	ERR	✓
word_puzzle	×	×	✓	×	ERR	×	✓
depth6_topo_sort	×	×	×	×	×	ERR	×
Total pass	3/8	3/8	6/8	3/8	4/8	3/8	4/8

errors with formatting and compilation discipline; our compile-reject taxonomy partially separates these factors but not perfectly. Finally, statistical conclusion validity is constrained by sample size: many confidence intervals span 30–50 percentage points, and claims are therefore directional unless corroborated by the higher-sample long-context runs.

11 Tool-Call Hallucination and the Case for Natively Trained RLMs

A persistent failure mode across our benchmarks is *tool-call hallucination*: the underlying LLM generates fabricated tool invocations instead of task-relevant answers. In our latest full benchmark run (4,151 total failures), we observed MiniMax-M2.5 emitting XML-encoded pseudo-tool-calls such as `<minimax:tool_call>` and `<FunctionCall>` tags directed at non-existent tools (`filesystem_search_files`, `inspect_context`, etc.). These hallucinated calls appeared in 0.07% of runs as explicit format leaks, but contributed more broadly to wrong-answer and empty-output categories when the model substituted tool attempts for direct reasoning.

	M2.1	M2.5	Q3.5	Q235	DSV3	K2	K2.5*
trap_2	✓	✓	✓	✓	✓	✓	✓
trap_1	✓	✓	✓	×	✓	✓	✓
match_1	×	✓	×	✓	×	×	×
match_2	✓	×	✓	×	×	×	×
state_tr	×	×	✓	×	✓	✓	×
deleg.	×	×	✓	✓	✓	ERR	✓
w_puz	×	×	✓	×	ERR	×	✓
topo_s	×	×	×	×	×	ERR	×
Total	3/8	3/8	**6/8**	3/8	4/8	3/8	4/8

Figure 13: Per-task pass/fail heatmap in depth1-iter3 mode across seven models. Green = pass, red = fail, orange = engine error. Tasks are ordered by decreasing solve rate; a clear difficulty gradient emerges from universally solved (top) to universally failed (bottom).

Table 13: Composite scores (quality × 0.55 + reliability × 0.25 + efficiency × 0.20) for best mode per model.

Model	Best mode	Pass%	Composite	Latency (s)	Readiness
Qwen3.5-397B	depth1-iter3	75.0	**76.1**	11.9	Promising
MiniMax-M2.5	no-rec-best3	50.0	65.9	13.4	Experimental
MiniMax-M2.1	depth1-iter3	37.5	64.5	7.4	NotProdReady
DeepSeek-V3.2	depth1-iter3	50.0	59.5	50.8	Experimental
Qwen3-235B-Think	depth1-iter3	37.5	58.6	36.4	NotProdReady
Kimi-K2-Think	depth1-iter3	37.5	51.5	71.7	NotProdReady
Post-FINAL-fix:					
Kimi-K2.5	default	62.5	83.7	39.1	Promising
Kimi-K2.5	no-rec-best3	75.0	79.7	38.1	Promising

This failure is not a scaffold bug but a *model training mismatch*: current models are trained to use tool-calling conventions from their native providers (OpenAI function calling, Anthropic tool use, etc.) but have never seen the RLM REPL interface during training. When placed in a recursive scaffold, they sometimes revert to their trained tool-use patterns rather than following the RLM prompt contract.

11.1 Evidence from Failure Taxonomy

Three categories in our failure taxonomy trace directly to this mismatch. In the `format_trace_or_scratchpad_leak` category (0.07% of runs), the model returns raw tool-call XML as its final answer—for example, emitting `<FunctionCall>` blocks directed at a nonexistent `inspect` tool. In the `missing_numeric_answer` category (0.07%), the model substitutes a `<minimax:tool_call>` invocation for the expected numeric output, attempting to call sandbox functions that do not exist. In the `hallucinated_when_should_abstain` category (0.05%), tasks requiring explicit abstention instead elicit tool calls to `filesystem_list_allowed_directo`. Although these categories are small in percentage terms, they represent a qualitatively distinct fail-

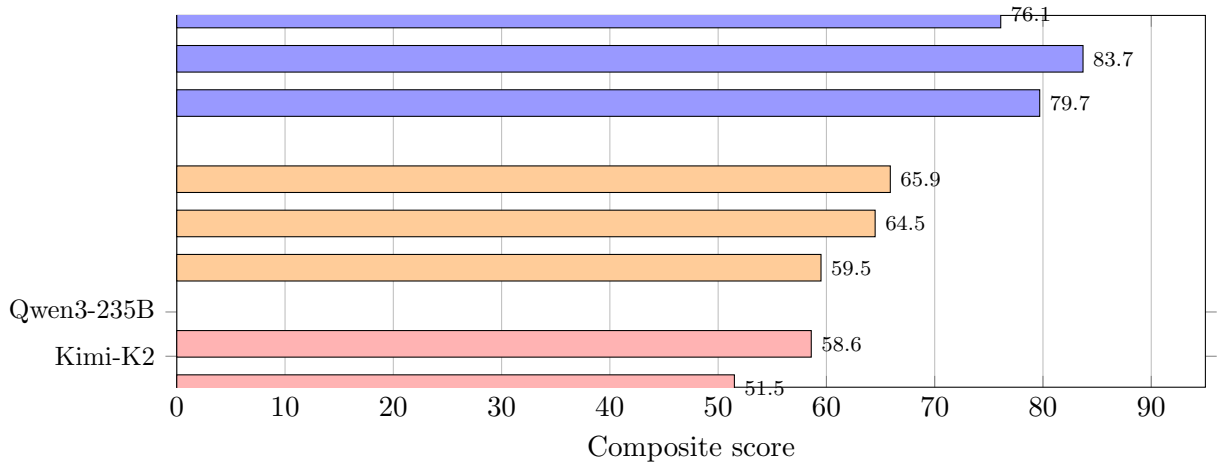

Figure 14: Composite readiness scores (best mode per model). Colours indicate readiness tier: blue = Promising (>70), orange = Experimental (60–70), red = NotProdReady (<60). Kimi-K2.5 (post-fix) achieves the highest composite at 83.7; Qwen3.5-397B is the strongest pre-fix model at 76.1.

Table 14: Bayesian posteriors for pass rate (selected configurations). Full posteriors available in supplementary.

Model	Mode	k/n	Post. mean	95% HDI	
Qwen3.5	depth1-iter3	6/8	0.700	0.400	0.925
Qwen3.5	no-rec-best3	1/8	0.200	0.028	0.483
DSV3.2	depth1-iter3	4/8	0.500	0.212	0.788
DSV3.2	no-rec-best3	3/8	0.400	0.137	0.701
K2.5*	no-rec-best3	6/8	0.700	0.400	0.925
K2.5*	default	5/8	0.600	0.299	0.863

ure mode: the model is not attempting the task at all, but rather invoking infrastructure that does not exist in the RLM environment.

11.2 Mitigation: Output Filtering

As an engineering mitigation, we implemented a post-processing step that strips hallucinated tool-call patterns from final answers using regex-based filtering. This removes `<minimax:tool_call>`, `<FunctionCall>`, and `<invoke>` XML blocks before answer extraction. While effective at preventing garbage output, this is a symptom-level fix that does not address the underlying training gap.

11.3 Toward Natively Recursive Models

The principled solution is to train models that natively understand the RLM interface. The updated RLM paper (v2) [15] provides the first evidence that this is both feasible and highly effective. Zhang et al. post-trained **RLM-Qwen3-8B** by distilling only ∼1,000 filtered RLM trajectories from a larger model (Qwen3-Coder-480B-A35B) evaluated on LongBenchPro [1] tasks. The resulting model outperforms the base Qwen3-8B by 28.3% on average across long-context tasks and

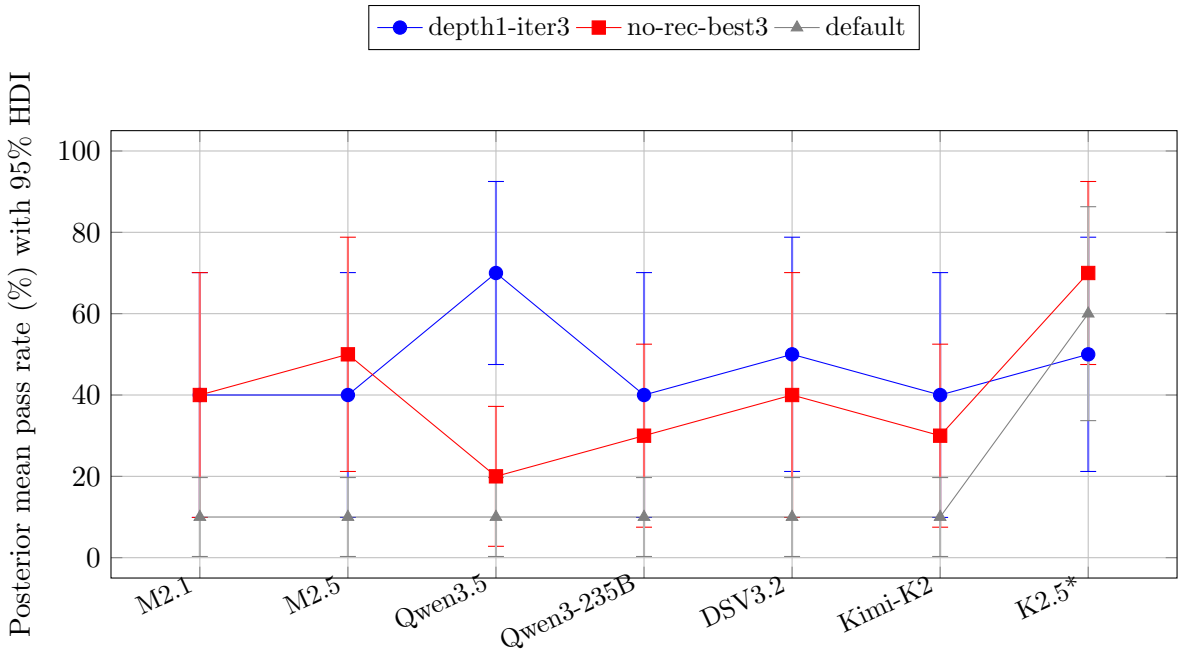

Figure 15: Bayesian 95% HDI for pass rate by model and mode. Points show posterior mean; bars show 95% credible interval under Beta(1,1) prior. For pre-fix models, default (gray) clusters at 10% while scaffolded modes separate. K2.5 (post-fix) shows overlapping intervals across all modes.

approaches vanilla GPT-5 quality on three benchmarks, despite a remarkably small training budget of 300 steps with batch size 64 using the prime-rl library [8]. Crucially, training focused on improving the root model's ability to interact with the REPL and to discern when sub-calls are useful; sub-call behaviour generalises from general-purpose reasoning capability rather than requiring explicit sub-call supervision. A programmatic trajectory correction step proved critical: 16% of collected trajectories contained incorrect `FINAL` usage and 13% incorrectly called REPL variables as final answers, and fixing these template errors substantially improved the distilled model's performance.

This result directly addresses our observed hallucination failures. A natively trained RLM would not revert to provider-specific tool-call conventions because it has learned the correct REPL interaction patterns. The trajectory correction finding is particularly relevant to Axon: it demonstrates that even large frontier models make systematic template mistakes in RLM usage, and that cleaning these during training data preparation is essential.

For Axon specifically, two paths forward are most promising. The first is trajectory collection and distillation: collecting RLM trajectories from strong scaffold runs (for example, using GPT-5 or Qwen3-Coder as the root model), filtering and correcting template errors, and distilling the result into a smaller model following the recipe in [15]. Axon's existing trajectory logging infrastructure makes this directly applicable. The second path is reinforcement learning from scaffold feedback, in which scaffold-level signals—pass/fail outcomes, depth utilisation, and format compliance—serve as reward signals for RLHF or DPO fine-tuning, building on the STaR bootstrapping approach [14, 13]. Axon's benchmark harness already produces the per-task scoring artefacts needed for this pipeline.

Table 15: Thompson sampling selection probability per model (10,000 rounds). Bold indicates the mode most frequently selected as optimal.

Model	default	depth1-iter3	depth6-single	no-rec-best3
M2.1	0.010	**0.490**	0.009	0.491
M2.5	0.005	0.318	0.004	**0.672**
Qwen3.5	0.001	**0.991**	0.001	0.008
Qwen3-235B	0.020	**0.666**	0.016	0.298
DSV3.2	0.005	**0.676**	0.005	0.314
Kimi-K2	0.020	**0.666**	0.016	0.298
Post-FINAL-fix:				
K2.5*	0.205	0.069	0.207	**0.519**
Macro-avg (6 pre-fix)	0.038	**0.554**	0.037	0.372

Table 16: Latency variability for selected configurations (seconds). Low CV indicates stable response times.

Model	Mode	Mean	Std	CV	Min	Max
Qwen3.5	depth1-iter3	11.9	5.2	0.44	5.5	23.4
Qwen3.5	default	11.4	6.5	0.57	4.3	23.5
M2.1	depth1-iter3	7.4	3.6	0.48	3.1	14.4
DSV3.2	default	31.5	10.4	0.33	12.3	50.4
Kimi-K2	depth6-single	23.0	22.7	0.98	7.1	72.5
K2.5*	depth1-iter3	28.6	23.3	0.81	8.4	86.2

12 Future Work: Plan Annealing and Prompt-Policy Learning

Our results point to two concrete directions for improving recursive scaffold performance. The first is plan annealing: explicit progress propagation across recursive calls so that child models receive only the unresolved objectives from a structured plan, rather than the entire problem context.

```
PLAN = [{task, state}] # state in {pending, in_progress, done, blocked}
for each recursive handoff:
  update PLAN using newly verified evidence
  ACTIVE = [item for item in PLAN if item.state != done]
  child_input = {active_tasks: ACTIVE, constraints, failure_notes, key evidence}
  child_output = sub_llm(child_input)
  merge child_output states back into PLAN
```

This mechanism is aligned with emerging evidence that stable internal structure, not only depth, matters for long reasoning quality [2]. Evaluation should test whether plan annealing lowers token use per solved task, reduces timeout/error rate, and improves strict executable pass rate at fixed budgets.

In parallel, prompt-policy optimization should be treated as a first-class scaffold optimization layer rather than ad-hoc prompt edits. We now expose policy bundles and depth gates as explicit knobs in both CLI and MCP, which enables offline policy search over benchmark artifacts. A practical optimization loop is to score candidate policies on pass rate, format-leak rate, and cost, then promote Pareto-dominant policies per model/mode family. This is directly compatible with

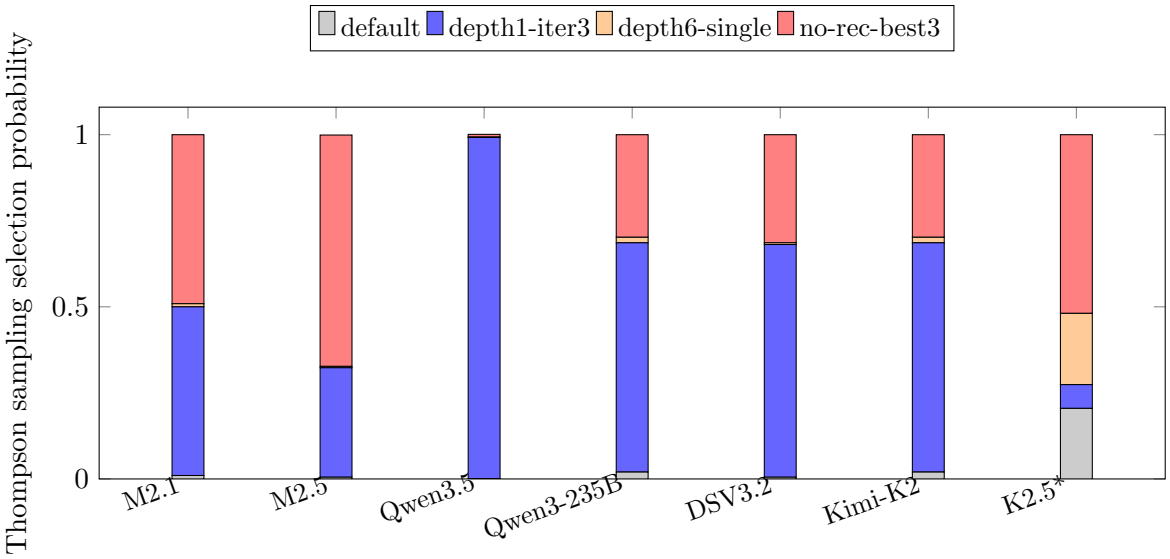

Figure 16: Thompson sampling mode allocation per model. Concentrated bars indicate high confidence in a single mode; diffuse bars indicate mode-insensitivity. Qwen3.5 is $> 99\%$ concentrated on depth1-iter3; K2.5 (post-fix) is nearly uniformly distributed. Across pre-fix models, attempt-rich modes (blue + red) capture $> 96\%$ of selections.

LLM-driven prompt optimization methods such as OPRO [17], automatic prompt engineering [16], and iterative self-improvement pipelines [14]; for decomposition-aware tasks, policy search can also borrow branch/search structure from Tree-of-Thought style methods [11].

13 Conclusion

This paper reframes Axonevaluation around a practical question arising from RLMdeployment: whether recursion provides measurable benefit over extra iterations on short-horizon tasks. A multi-model Pareto analysis across six pre-fix models and one post-fix model (Kimi-K2.5) on the Synthetic platform (224 total runs) yields three principal conclusions.

The most immediate lesson is that scaffold bugs can masquerade as model limitations. A FINAL-extraction bug in the fallback path caused depth6-single-pass to score 0% across all pre-fix models, even when the underlying answers were correct; after fixing, Kimi-K2.5 achieved 62.5% in depth6-single-pass, demonstrating that deep recursion can work when the scaffold faithfully surfaces model outputs. This finding underscores the importance of end-to-end contract verification before attributing failure to model capability.

At the deployment level, the Pareto frontier reveals two co-dominant configurations: Qwen3.5-397B with depth1-iter3 (75% pass, 11.9 s, \$0.0044 per task) offers the best speed and cost, while Kimi-K2.5 with no-rec-best3 (75% pass, 38.1 s, \$0.0064 per task) provides the strongest single-pass robustness at 62.5% even in default mode. The tenfold speed variation across models (7.4 s to 71.7 s) makes latency a first-order deployment concern that interacts with mode selection.

Perhaps most consequentially, whether recursion or iteration wins is model-dependent. Qwen3.5 strongly favours recursion (75% versus 12.5% for no-rec-best3), MiniMax-M2.5 favours iteration (50% versus 37.5%), and Kimi-K2.5 is relatively mode-insensitive (50–75% across all modes). This interaction effect between model capability and scaffold mode cautions against universal mode

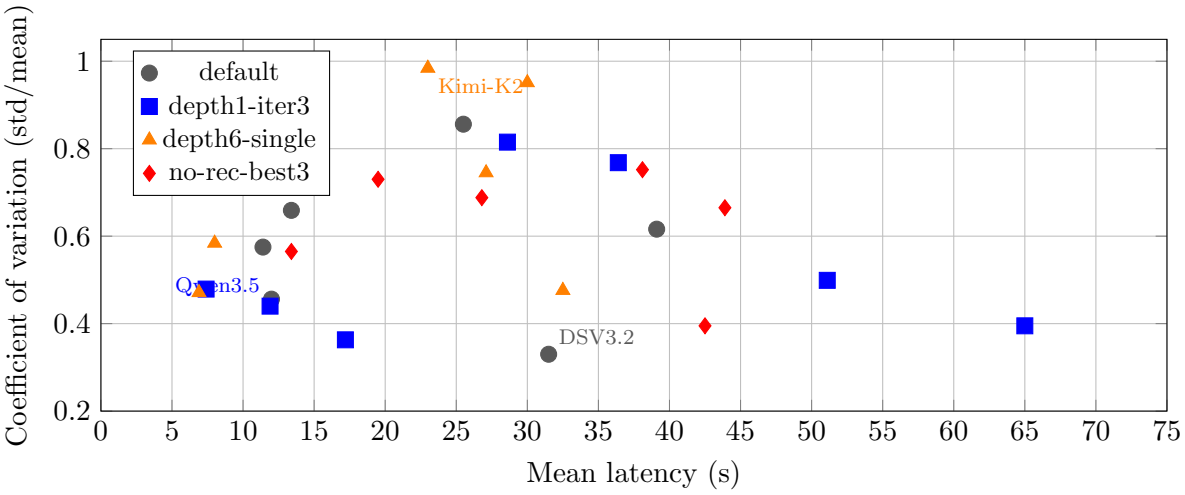

Figure 17: Latency mean vs coefficient of variation by model and mode. Ideal configurations are in the lower-left quadrant. Qwen3.5 with depth1-iter3 (blue, 11.9s, CV=0.44) combines low latency with moderate stability. Deep single-pass modes (orange) tend toward high CV.

recommendations and motivates the adaptive policy selection mechanisms described in our future work.

References

[1] Jiayu Chen et al. LongBenchPro: Long-context benchmark for professional tasks, 2026. Used as training data source for RLM-Qwen3-8B trajectories.

[2] Qiguang Chen et al. The molecular structure of thought: Mapping the topology of long chain-of-thought reasoning. 2026.

[3] Kelly Hong, Anton Troynikov, and Jeff Huber. Context rot: How context degradation affects llm performance, 2025. Accessed 2026-02-24.

[4] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? 2024.

[5] Yubin Kim et al. Towards a science of scaling agent systems. 2025.

[6] Yubin Kim and Xin Liu. Towards a science of scaling agent systems: When and why agent systems work, 2026. Google Research Blog, Accessed 2026-02-25.

[7] METR. RE-bench: Measuring ability to complete long tasks, 2025. Accessed 2026-02-24.

[8] Prime Intellect. prime-rl: Reinforcement learning library, 2025. Used for RLM-Qwen3-8B fine-tuning.

[9] Prime Intellect Research. Recursive language models: the paradigm of 2026, 2026. Accessed 2026-02-25.

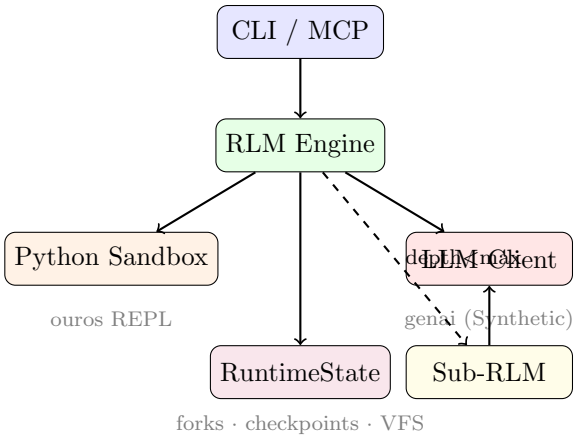

Figure 18: Axon architecture. The RLM engine orchestrates iteration loops over a Python sandbox (ouros) and LLM client (genai), with RuntimeState managing fork/checkpoint/VFS state. When depth < max_depth, `llm_query()` spawns a child RLM with its own sandbox.

[10] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. 2024.

[11] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. 2023.

[12] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. 2023.

[13] Eric Zelikman, Georges Harik, Yijia Shiv, Jesse Mu, and Noah D. Goodman. Quiet-STaR: Language models can teach themselves to think before speaking. 2024.

[14] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Self-taught reasoner bootstrapping reasoning with reasoning. 2022.

[15] Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models. 2025. v2 introduces RLM-Qwen3-8B, the first natively recursive language model.

[16] Yongchao Zhou, Aman Madaan, Sai Potharaju, Ashma Gupta, Stephen McAleer, et al. Large language models are human-level prompt engineers. In *NeurIPS 2022 Workshop on Foundation Models for Decision Making*, 2023.

[17] Yongchao Zhou, Aman Madaan, Sai Potharaju, Ashma Gupta, Stephen McAleer, Jason Pombra, Sunita Jain, Ruslan Salakhutdinov, Abhinav Gupta, Diyi Yang, et al. Large language models as optimizers. 2023.