λ-RLM: STATICALLY RESTRICTED CONTROL FOR RECURSIVE LONG-CONTEXT REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Long context is not synonymous with reliable reasoning: relevant evidence is easily lost, while standard Recursive Language Models (RLMs) address the problem by asking the model to generate and execute its own control code. This open-ended strategy extends effective context, but introduces malformed programs, unpredictable cost, and uncontrolled recursion. We introduce λ-RLM, a structured alternative that expresses recursive reasoning as a closed λ-calculus program over a compact library of compositional operators. The language model is invoked only on bounded leaf subproblems; decomposition, filtering, traversal, and aggregation follow a fixed plan that can be inspected before execution. This separation guarantees termination, bounds the number of model calls (leaves, plus one per internal node under neural composition) and hence their cost, and supports task-appropriate accuracy analyses for both conjunctive and retrieval settings. Across five long-context benchmarks and nine open-weight models, five-seed comparisons, λ-RLM improves accuracy most strongly for weak and medium models while reducing latency by $4.0\times$ on average and up to $6.5\times$. These results show that explicit functional control can make recursive long-context reasoning faster, more reliable, and easier to analyze without increasing model scale.

1 INTRODUCTION

Long-context reasoning is limited both by finite context windows and by accuracy degradation within those windows, including lost-in-the-middle effects (Liu et al., 2023; Wang et al., 2024). Inference-time decomposition addresses both problems by solving bounded subproblems and composing their outputs (Zhou et al., 2023; Yao et al., 2023; Yang et al., 2025).

Recursive Language Models (RLMs) operationalize decomposition by placing the prompt in a read–eval–print loop (REPL) and asking the model to write arbitrary Python that inspects, partitions, and recursively queries it (Zhang et al., 2026). This flexibility introduces failures unrelated to semantic reasoning: malformed code, unbounded or duplicated recursion, invalid intermediate formats, and unpredictable cost.

We propose λ-RLM. The prompt remains external, but control is restricted to typed, pre-verified operators such as SPLIT, MAP, FILTER, and REDUCE. The model is a bounded leaf oracle; a fixed plan determines decomposition and composition. We use λ-calculus as a *specification language*: after planning, the controller is a closed term whose control flow cannot be changed by model output. The same guarantees could be implemented by an equivalent verified higher-order function; the measured gains arise from static restricted control rather than from notation. Recursion is represented by a fixed-point operator in the specification and implemented as an ordinary strict recursive function; termination follows from the fact that every recursive call processes a strictly smaller prompt.

Our contributions are: *(i)* a finite typed controller separating neural semantics from symbolic structure; *(ii)* termination, tight cost, fan-out[1] planning, and task-appropriate accuracy analyses; *(iii)* five-seed experiments on nine models and five benchmarks, including assumption checks, correlation measurements, planner sensitivity, a structured-planning baseline, and factorized ablations; and *(iv)* an anonymous implementation and evaluation artifact. The system is a partial substitute for model

[1] Fan-out k is the number of chunks SPLIT cuts a prompt into at each recursion level.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

scale in the weak and medium regimes, but its advantage narrows for frontier models and open-ended code/search tasks.

2 Related Work

Long-context scaling. Truncation and sliding windows discard evidence needed for global consistency (Liu et al., 2023; Wang et al., 2024). Retrieval, decomposition, and reconstruction methods instead select or process bounded pieces (Jin et al., 2024; Li et al., 2024; Xu et al., 2026; Chen et al., 2026). λ-RLM focuses on making the resulting execution structure fixed and auditable.

Recursive and hierarchical reasoning. Least-to-Most, Tree-of-Thoughts, planning agents, and RLM all decompose inference (Zhou et al., 2023; Yao et al., 2023; Zhang et al., 2026). Uncertainty-aware program search retains model-authored control (Alizadeh et al., 2026); λ-RLM instead fixes the admissible control language before execution. Section 6 includes a one-shot plan-generation baseline to separate structured planning from restricted execution.

Agentic and neuro-symbolic systems. Open-ended agents exhibit control-flow and coordination failures (Zhu et al., 2025; Cemri et al., 2025). Formal calculi can specify agent interactions and information flow (Garby et al., 2026; Bhardwaj, 2026). Our use is deliberately narrower: the calculus specifies a static recursive controller, while task plans remain human-designed.

3 Notation and Background

We provide the minimal λ-calculus machinery needed for λ-RLM; a full prose primer with worked examples is in Appendix A. We consider the untyped λ-calculus, where expressions e are defined inductively as variables x, abstractions $\lambda x.\, e$ (anonymous functions), and applications $e_1\, e_2$ (function application). The operational semantics is given by β-reduction:

$$(\lambda x.\, e)\, a \xrightarrow{\beta} e[x := a],$$

where $e[x := a]$ denotes substitution of a for x in e. We adopt the standard convention that application associates to the left, i.e., $f\, a\, b \equiv (f\, a)\, b$.

Definition 1 (Fixed-Point Operator)**.** *A fixed-point operator* ***fix*** *satisfies* $\textbf{\textit{fix}}(g) = g(\textbf{\textit{fix}}(g))$*. We use* ***fix*** *only to express the recursive controller as a closed λ-term. In the implementation it is realized by an ordinary recursive function, and termination is established separately by proving that prompt size strictly decreases at every recursive call (Theorem 1).*

Definition 2 (Base Language Model)**.** *A base language model is a (possibly stochastic) mapping, $\mathcal{M} : \Sigma^* \to \Delta(\Sigma^*)$, where $\Delta(\Sigma^*)$ denotes the space of probability distributions over sequences. The model has a context window $K \in \mathbb{N}$, and its behavior is well-defined only for inputs of length $|P| \leq K$.*

Definition 3 (Cost Function)**.** *The cost of invoking $\mathcal{M}$ on n tokens:*

$$\mathcal{C}(n) = c_{in} \cdot n + c_{out} \cdot \bar{n}_{out} \tag{1}$$

where c_{in}, c_{out} are per-token prices and $\bar{n}_{out}$ is expected output length.

Definition 4 (Direct-inference decay model)**.** *For comparison only, we model direct-inference accuracy at length n as*

$$\mathcal{A}(n) = \mathcal{A}_0 \cdot \rho^{\,n/K}, \quad \rho \in (0, 1] \tag{2}$$

where $\mathcal{A}_0$ is peak accuracy and ρ is a fitted context-rot factor. This is a simplifying reference model, not a universal law: relevant added context can improve a particular instance, and the theoretical guarantees below do not require this exact functional form.

Definition 5 (Composition Operator)**.** *A composition operator $\oplus$ merges child outputs into one output: $\oplus : \Sigma^* \times \Sigma^* \to \Sigma^*$ for string-valued tasks, while for pairwise tasks the merged output is a set of index pairs serialised as a string. We define a family $\{\oplus_t\}_{t \in \mathcal{T}}$ indexed by task type t. For tasks such as* SEARCH *and* AGGREGATE*, $\oplus_t$ is a deterministic symbolic function. For tasks such as* SUMMARISE*, $\oplus_t = \mathcal{M} \circ$ CONCAT *involves a neural call and is therefore stochastic; in this case $\mathcal{A}_\oplus$ (Assumption A5 in Section 5) captures the probability that $\oplus$ preserves a correct answer.*

Algorithm 1 RLM (Zhang et al., 2026)

Input: prompt P
Output: response Y

state ← InitREPL(`prompt`=P)
state ← AddFunction(state, `sub_RLM`)
hist ← [Metadata(state)]
while True do
  code ← $\mathcal{M}$(hist) ▷ Δ_1
  (state, stdout) ← $\mathcal{E}$(state, code) ▷ Δ_2
  hist ← hist ‖ code ‖ Metadata(stdout)
  if state[`Final`] is set **then**
    return state[`Final`]
  end if
end while

Algorithm 2 λ-RLM (ours)

Input: prompt P
Output: response Y

state ← InitREPL(`prompt`=P)
state ← AddFunction(state, `sub_`$\mathcal{M}$)
state ← RegisterLibrary(state, $\mathcal{L}$) ▷ Δ_1
t_{type} ← $\mathcal{M}$(menu ‖ meta)
if $|P| \leq K$ **then**
  return `sub_`$\mathcal{M}(P)$
end if
$(\oplus, \pi, k^*, \tau^*)$ ← Plan($t_{\text{type}}, |P|$) ▷ Δ_2
Φ ← BuildExecutor($k^*, \tau^*, \oplus, \pi$) ▷ Δ_3
return $\Phi(P)$

Figure 1: RLM vs. λ-RLM. Blue: model calls. Orange: REPL code, written by the model every turn in RLM (Δ_1, Δ_2), a pre-verified combinator chain in λ-RLM (Δ_3). Green: deterministic planning, absent from RLM (Δ_1, Δ_2). λ-RLM has no model-driven loop; Φ calls the model only at leaves.

Example. For child outputs s_1 = `"desc:3, num:1"` and s_2 = `"desc:2, num:4"`, `aggregate` ($\oplus$ = Merge) adds counts per label and returns `"desc:5, num:5"`; `search` (FilterBest) keeps the non-`"no match"` output. Both are deterministic: no model call. `summarise` ($\mathcal{M} \circ$ Concat) asks the model to summarise the concatenation, so its output can vary across runs. A longer trace is in Appendix B.3.

4 The λ-RLM Framework

The key idea is to preserve RLM's prompt-as-environment abstraction while replacing model-authored control with a small trusted runtime. The model handles semantic leaf subproblems; deterministic combinators handle traversal, filtering, and aggregation. A worked execution trace appears in Appendix B.3.

4.1 From Open-Ended Control to a Restricted Runtime

Standard RLM operates through a REPL-style loop where the model iteratively generates executable code based on the interaction history, executes it in the REPL, and appends the resulting output to its history. The REPL also exposes a recursive sub-RLM function, allowing the model to invoke itself on subproblems. This process continues until the model sets a final response, which is then returned.

This delegates control entirely to a stochastic model; it must decide what to inspect, how to decompose, when to recurse, and when to stop creating an open-ended loop with no termination guarantee and a hard dependency on coding ability. λ-RLM retains the REPL abstraction but restricts what may execute inside it. The environment still stores the prompt externally and supports recursive sub-calls to the base model; what changes is the control interface. Rather than synthesizing arbitrary programs token by token, execution is restricted to a small typed library of trusted combinators. This isolates the source of uncertainty: the model contributes only at leaf subproblems, while decomposition, traversal, filtering, and aggregation are handled by deterministic operators with verifiable behavior. Once the decomposition rule and base threshold are fixed, the depth of recursion, the number of model calls, and the overall cost become explicit functions of the input size.

A Compact Combinator Library. We design our combinator library to be minimally sufficient for the kinds of recursive control patterns that repeatedly arise in long-context reasoning. In particular, such tasks typically require only a small set of operations: partitioning an input into manageable

pieces, selectively inspecting or pruning those pieces, applying a subroutine to each retained component, and aggregating the resulting outputs into a final answer. We therefore choose a library of typed, deterministic combinators that correspond exactly to these roles. This choice is deliberate: the goal of λ-RLM is not to maximise expressivity at the control level, but to retain only the expressivity needed for structured decomposition while eliminating the open-ended failure modes of free-form code generation. Table 1 gives the library and six plans. Here Σ^* denotes strings, $[\alpha]$ lists over type α, $\alpha \to \beta$ functions, and $\times$ product arguments; for example, $\textsc{Map} : (\alpha \to \beta) \times [\alpha] \to [\beta]$ maps a function over a list. A *task type* $t \in \mathcal{T}$ is one of six menu labels, an *execution plan* π is a well-typed combinator chain, and $\oplus$ is the fold used to merge child outputs. The finite term $\textsc{PlanOf} : \mathcal{T} \to (\oplus, \pi)$ encodes Panel B. Plans are human-designed once; λ-calculus does not discover them.

Table 1: A compact combinator library $\mathcal{L}$ and examples of task-specific execution plans.

Panel A: Combinators (pre-verified, loaded into REPL)

Combinator	Type Signature	Description
Split	$\Sigma^* \times \mathbb{N} \to [\Sigma^*]$	Partition a string into k contiguous chunks
Peek	$\Sigma^* \times \mathbb{N}^2 \to \Sigma^*$	Extract a substring by start and end position
Map	$(\alpha \to \beta) \times [\alpha] \to [\beta]$	Apply a function from $(\alpha \to \beta)$ to every element of a list $[\alpha]$
Filter	$(\alpha \to \mathbb{B}) \times [\alpha] \to [\alpha]$	Retain elements satisfying a predicate from $\mathbb{B}$
Reduce	$(\beta \times \beta \to \beta) \times [\beta] \to \beta$	Fold a list into a single value via a binary operator
Concat	$[\Sigma^*] \to \Sigma^*$	Join a list of strings into one string
Cross	$[\alpha] \times [\beta] \to [(\alpha, \beta)]$	Cartesian product of two lists
$\mathcal{M}$	$\Sigma^* \to \Sigma^*$	*Neural oracle*: invoke the base model on a sub-prompt

Panel B: Task type, composition operator, and execution plan

Task type	Composition $\oplus$	Execution plan π
`search`	FilterBest	Split $\to$ Map(Peek) $\to$ Filter $\to$ Map($\mathcal{M}$) $\to$ Best
`classify`	Concat	Split $\to$ Map($\mathcal{M}$) $\to$ Concat
`aggregate`	Merge	Split $\to$ Map($\mathcal{M}$) $\to$ Merge
`pairwise`	Cross $\circ$ Filter	Split $\to$ Map($\mathcal{M}$) $\to$ Parse $\to$ Filter $\to$ Cross
`summarise`	$\mathcal{M} \circ$ Concat	Split $\to$ Map($\mathcal{M}$) $\to$ Concat $\to$ $\mathcal{M}$
`multi_hop`	$\mathcal{M} \circ$ Concat	$\text{Split}_\delta \to$ Map(Peek) $\to$ Filter $\to$ Map($\mathcal{M}$) $\to$ $\mathcal{M}_{\text{synth}}$

4.2 Core Formulation and Planner

λ-RLM is defined by the single lambda term:

$$\lambda\text{-RLM} \equiv \texttt{fix}\Big(\lambda f.\, \lambda P.\, \textbf{if } |P| \le \tau^* \textbf{ then } \mathcal{M}(P) \textbf{ else } \textsc{Reduce}\big(\oplus,\ \textsc{Map}(\lambda p_i.\, f\, p_i,\ \textsc{Split}(P, k^*))\big)\Big), \tag{3}$$

where P is the prompt, k is fan-out, τ the leaf threshold, and $\oplus$ the task-specific composition. This is the identity-hook case of BuildExec (Eq. 4). The fixed controller supplies the recursive variable f; the model never decides whether to recurse.

The planner has two separable parts. A single menu-selection call predicts t, after which $\textsc{PlanOf}(t)$ returns the fixed $(\oplus, \pi)$. Numeric planning is deterministic. Let $k_{\max} = \lfloor K/\bar{n}_{\text{out}} \rfloor$ be the largest fan-in whose composed outputs fit the context window. For each integer $2 \le k \le k_{\max}$, the planner sets $\tau = K$, $d = \lceil \log_k(n/K) \rceil$, evaluates the exact cost in Proposition 4, and chooses the least-cost feasible k. When one split suffices, $k_0 = \lceil n/K \rceil \le k_{\max}$, the unique optimum is $k^* = k_0$, $\tau^* = \lceil n/k^* \rceil$, and $d = 1$. Thus the 131K-token OOLONG example with $K = 32$K uses $k^* = 5$, while the 145-token example with $K = 50$ uses $k^* = 3$. For tasks with non-trivial structure, we provide specialized instantiations in Appendix A. The key insight for pairwise tasks: $O(n^2)$ pair computation is purely symbolic (zero neural cost), while only $O(n/K)$ classification calls are neural. For multi-hop search, preview-based filtering reduces the corpus before any expensive neural reading.

4.3 Compilation to a Closed Executor

Although task-specific plans are selected by finite dispatch, the contribution is not the dispatch table itself but a parameterized, closed controller. The dispatch mechanism selects $(\oplus, \pi)$, whereas

recursive semantics are inherited from BUILDEXEC. Termination and complexity are therefore established once for the generic controller and transfer by substitution. An equivalent verified higher-order function could provide the same property; the λ-term is our compact formal specification. Figure 3 illustrates the resulting execution pipeline on a representative OOLONG instance (131K tokens, $K = 32$K). The planner arithmetic producing (k^*, τ^*) is deterministic and precedes execution; runtime evaluation itself consists of three β-reduction stages.

Generic executor template. The central object is an open λ-term parameterized by $(k, \tau, \oplus, \pi)$:

$$\textsc{BuildExec} \equiv \lambda k.\, \lambda \tau.\, \lambda \oplus .\, \lambda \pi.\, \texttt{fix}\Big(\lambda f.\, \lambda P.\ \textbf{if}\ |P| \le \tau\ \textbf{then}\ \mathcal{M}\big(\pi_{\text{leaf}}(P)\big) \\ \textbf{else}\ \textsc{Reduce}\big(\oplus,\ \textsc{Map}(f, \pi_{\text{pre}}(\textsc{Split}(P, k)))\big)\Big), \tag{4}$$

where π_{leaf} specifies leaf-level prompt construction and π_{pre} optionally applies symbolic preprocessing. BUILDEXEC is verified once offline and constitutes the sole recursive component of λ-RLM.

Three-stage reduction. Execution proceeds in three stages. **S1 (plan retrieval).** Applying the task descriptor to the dispatch term PLANOF yields the corresponding composition operator $\oplus$ and preprocessing skeleton π by one β-reduction. **S2 (executor instantiation).** Substituting (k^*, τ^*) together with the retrieved $(\oplus, \pi)$ into BUILDEXEC produces a closed executor Φ. This stage is purely symbolic and introduces no neural evaluation. **S3 (recursive execution).** Applying Φ to prompt P unfolds the fixed point into a bounded recursion tree. Internal nodes perform symbolic partitioning and composition, while neural computation is restricted to leaves satisfying $|P| \le \tau^*$. The recursive structure is induced explicitly by the fixed-point operator. Since each recursive call strictly decreases prompt size under SPLIT, Theorem 1 guarantees termination after depth at most

$$d = \left\lceil \log_{k^*}(n/\tau^*) \right\rceil, \tag{5}$$

with no more than $(k^*)^d$ leaf evaluations. All three stages manipulate the same λ-term object. This shared representation enables Theorems 1–3 and Proposition 4 to apply uniformly across all task instantiations in Panel B of Table 1, without requiring task-specific correctness proofs.

5 THEORETICAL GUARANTEES

Let $n = |P|$, fan-out $k \ge 2$, leaf threshold $\tau \le K$, depth $d = \max\{0, \lceil \log_k(n/\tau) \rceil\}$, and leaf count $L \le k^d$. We use: (A1) $\mathcal{M}$ halts on inputs of length at most K; (A2) non-neural combinators are total and deterministic; (A3) costs are nonnegative; (A4) leaf accuracy $\mathcal{A}(\cdot)$ is non-increasing on $[1, K]$; (A5) $\mathcal{A}_\oplus$ lower-bounds correctness preservation at each neural composition level; (A6a) $\textsc{Split}(P, k)$ returns exactly k contiguous pieces with $|P_i| \le \lceil |P|/k \rceil$ (possibly empty trailing pieces); and (A6b) FILTER retains $k' \le k$ pieces. For conjunctive tasks, (A7c) joint leaf success is lower-bounded by the product of marginal successes. For localisable retrieval, (A7r) exactly one leaf contains the evidence and each non-evidence leaf independently produces a surviving false positive with probability at most p_{fp}.

Theorem 1 (Termination and call count)**.** *Under A1, A2, and A6, for every $P \in \Sigma^*$ with $|P| = n \in \mathbb{N}$, $\Phi(P)$ terminates. Its depth is at most d and it makes at most $L \le k^d$ leaf calls. Counting task-type selection, the complete system makes*

$$N(n) \le \begin{cases} k^d + 1 & \textit{symbolic } \oplus, \\ k^d + \dfrac{k^d - 1}{k - 1} + 1 & \textit{neural } \oplus\ (\oplus = \mathcal{M} \circ \textsc{Concat}), \end{cases} \tag{6}$$

where the middle term counts one composition call per internal node, whose input (the concatenated child outputs) is assumed to fit in K, i.e. $k \le k_{\max}$.

The proof is based on $r(P') = \lceil \log_k(|P'|/\tau) \rceil$: A6a strictly lowers rank, A6b cannot increase it, and A1–A2 terminate the leaf and symbolic cases; a neural $\oplus$ halts by A1 since its input fits in K. Full proofs are in Appendix C.

Theorem 2 (Tight cost expression). *For a full balanced tree, per-input-token cost c_{in}, mean leaf output length $\bar{n}_{\text{out}}$, per-output-token cost c_{out}, and linear internal composition cost $c_{\oplus}k$, the exact cost (up to fixed planner overhead) is*

$$T(n,k) = c_{\text{in}}n + c_{\text{out}}\bar{n}_{\text{out}}L + c_{\oplus}k\frac{L-1}{k-1}, \qquad L = k^d. \tag{7}$$

Filtering only decreases this value. If composition is symbolic, $c_{\oplus} = 0$; cost then depends on k only through the rounded leaf count L, and can be identical for several fan-outs.

Theorem 3 (Task-appropriate accuracy bounds). *Under A4–A7:* (i) *if $n \le \tau$ (equivalently $d = 0$), then $\mathcal{A}_{\lambda\text{-RLM}}(n) = \mathcal{A}(n) \ge \mathcal{A}(\tau)$;* (ii) *for conjunctive tasks,*

$$\mathcal{A}_{\lambda\text{-RLM}}(n) \ge \mathcal{A}(\tau)^L \mathcal{A}_{\oplus}^d; \tag{8}$$

thus the general bound retains an exponential term in $L = \Theta(n/\tau)$ whenever $\mathcal{A}(\tau) < 1$; (iii) *only in the limiting case $\mathcal{A}(\tau) = 1$ does the remaining $\mathcal{A}_{\oplus}^d$ term yield power-law decay in n; and* (iv) *for localisable retrieval,*

$$\mathcal{A}_{\lambda\text{-RLM}}(n) \ge \mathcal{A}(\tau)(1 - p_{\text{fp}})^{L-1}\mathcal{A}_{\oplus}^d. \tag{9}$$

With symbolic composition and $p_{\text{fp}} = 0$, the retrieval bound is constant in n.

Proposition 4 (Fan-out planning). *Let $2 \le k \le k_{\max}$, where $k_{\max}$ enforces composition fan-in. The cost-minimizing fan-out is the finite minimizer of equation 7; $k = 2$ is not generally optimal. If a one-level split is feasible, $k_0 = \lceil n/K \rceil \le k_{\max}$, then $k^* = k_0$, $\tau^* = \lceil n/k^* \rceil$, and $d^* = 1$. Otherwise the planner enumerates the admissible integers using the exact objective.*

The previous $k = 2$ conclusion results from replacing L by the loose surrogate nk/τ, which introduces an artificial term linear in k. The exact expression instead exposes the trade-off among rounded leaf count, output cost, and composition fan-in. Appendix D validates the one-level rule with a fan-out sweep.

6 Experiments

We compare λ-RLM against **(P1) Direct**, which processes the full prompt in one call (truncated or marked failed when $n > K$), and **(P2) RLM** (Zhang et al., 2026), which generates arbitrary Python within an open-ended REPL loop. We additionally test **(P4) PlanGen**, a one-shot LLM-generated decomposition plan, on S-NIAH. Both P2 and P3 (λ-RLM) store the prompt in the REPL; the distinction is execution mode: P2 synthesizes free-form code online, whereas P3 executes a single pre-compiled combinator chain. We evaluate three open-weight model families across weak/medium/strong scales (Table 8), served via vLLM with the same model used at root and leaves. Benchmarks span five tasks from (Zhang et al., 2026), covering constant-, linear-, quadratic-, and multi-hop complexities across context lengths from 8K to 11M tokens (Table 9). P2 uses the original RLM prompts. Each headline configuration is run with five seeds; we report means, paired tests at $\alpha = 0.05$, task accuracy/F1, wall-clock latency, and call count. Appendix D gives implementation, planner, and statistical details.

Main Quantitative Results and Efficiency-Accuracy Tradeoff

Table 3 reports the means. Paired five-seed tests yield **36 significant λ-RLM wins, eight ties, and one significant RLM win** among 45 model–task comparisons. Qwen3-8B improves from 11.4% to 31.1% (+19.7 points), and the largest task-level mean gain is +27.0 points on OOL-Pairs. Pooled latency falls by **4.0**×, reaching **6.2**× on OOL-Pairs (up to 6.5× in a single cell). The gains are not universal: eight cells with a higher RLM mean are statistically tied, and Codestral-22B on BrowseComp+ is the significant RLM win. Qwen3-8B λ-RLM attains 93% of Llama-70B RLM accuracy (31.1 vs. 33.4), rather than tying it, while running 3.2× faster.

Table 2: no baseline reported by Zhang et al. (2026) nears λ-RLM on OOLONG or OOL-Pairs; offloading agents lead on BC+ and RLM on CodeQA.

Scaling Across Model Capability and Task Complexity. Table 4 shows that the advantage is concentrated in weak and medium models (15/15 and 14/15 higher means) and narrows in the strong tier (7/15). Improvements are largest when neural classification can be separated from symbolic pair

Table 2: Other paradigms vs. λ-RLM (accuracy/F1, %). Top block: copied, not re-run, from Zhang et al. (2026) (full version in Appendix D.3). Bottom: our means on Qwen3-235B; backbones differ, so cross-block gaps are indicative (like-for-like: Table 3). *: context limits.

Method	CodeQA	BC+	OOLONG	OOL-Pairs
Reported by Zhang et al. (2026), Qwen3-Coder-480B				
Base model	20.0*	0.0*	36.0	0.1
CodeAct (+ BM25)	24.0*	12.7	38.0	0.3
CodeAct (+ sub-calls)	26.0*	0.0*	32.0	0.1
Compaction agent	50.0	38.0	44.1	0.3
OpenCode	12.0*	0.0*	36.0	0.0
OpenCode (+ ctx. offloading)	40.0	**58.0**	24.0	2.1
RLM (depth = 1)	56.0	44.7	48.0	23.1
Ours, Qwen3-235B, five seeds				
RLM	**58.6**	32.8	56.5	38.4
λ-RLM	54.2	42.1	**68.4**	**61.5**

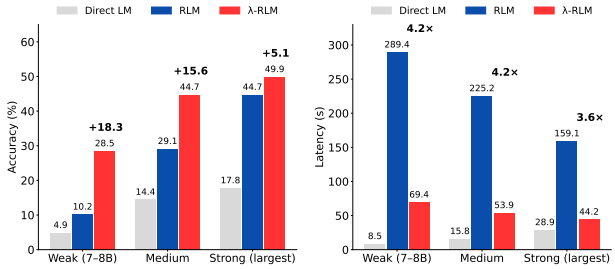


Figure 2: Accuracy and latency by model tier, averaged over the tier's models and five tasks (from Table 3). The weak-tier speedup is $4.2\times$; pooled over all 45 cells it is $4.0\times$, and the largest single-cell speedup is $6.5\times$.

Table 3: **Accuracy (%) / Latency (s)** across all models, tasks, and paradigms. Bold = best accuracy per cell group. For BrowseComp+ (6–11M tokens), P1: Direct scores 0.0^* everywhere as input far exceeds all context windows. Among recursive paradigms, λ-RLM is always faster in every cell.

		Qwen3			Llama			Mistral		
Task	**Paradigm**	**8B**	**32B**	**235B**	**8B**	**70B**	**405B**	**7B**	**8x22B**	**Cdsrl**
S-NIAH	P1: Direct	3.2/12.3	18.4/18.7	31.7/42.1	4.1/11.8	22.6/21.4	35.2/48.6	2.8/10.5	19.1/20.2	14.3/16.8
	P2: RLM	8.4/164.3	28.3/142.8	46.8/98.6	10.2/178.2	31.5/128.4	**52.4**/86.3	6.1/195.7	26.4/155.1	29.7/120.4
	P3: λ-RLM	**24.6**/45.9	**41.8**/38.2	**51.3**/31.4	**22.1**/48.7	**44.2**/35.6	49.8/28.1	**18.5**/52.3	**38.6**/41.8	**40.2**/36.5
OOLONG	P1: Direct	12.5/8.4	31.2/14.2	44.0/35.8	14.3/7.9	34.8/16.1	47.1/40.3	10.8/7.1	28.5/15.4	22.6/12.6
	P2: RLM	24.1/241.6	42.7/198.3	56.5/125.7	22.6/258.4	45.3/182.5	**63.8**/108.2	18.3/284.1	38.9/215.3	**48.2**/168.7
	P3: λ-RLM	**48.3**/62.4	**62.5**/51.7	**68.4**/42.3	**45.7**/66.8	**61.2**/48.2	61.7/38.5	**40.2**/71.5	**55.8**/56.4	45.6/48.1
OOL-Pairs	P1: Direct	0.1/6.2	0.3/10.8	0.1/28.4	0.1/5.8	0.4/12.3	0.2/32.1	0.0/5.1	0.2/11.7	0.1/9.4
	P2: RLM	4.2/312.5	18.6/264.7	38.4/178.3	3.8/338.1	21.3/241.8	42.7/156.4	2.1/365.2	14.5/288.6	22.8/224.5
	P3: λ-RLM	**34.8**/48.6	**48.2**/41.2	**61.5**/34.7	**31.6**/52.1	**50.7**/38.4	**64.3**/30.8	**28.4**/55.8	**44.1**/44.6	**47.6**/38.2
BrowseComp+	P1: Direct	0.0/3.1	0.0/5.8	0.0/14.2	0.0/2.9	0.0/6.4	0.0/16.8	0.0/2.6	0.0/6.1	0.0/4.8
	P2: RLM	1.8/425.6	16.2/358.2	32.8/248.7	2.4/448.3	22.6/325.4	**38.4**/215.6	1.2/482.1	14.8/372.5	**28.5**/298.4
	P3: λ-RLM	**12.4**/95.4	**28.5**/78.3	**42.1**/62.5	**14.2**/102.6	**35.8**/72.8	36.2/55.4	**10.6**/108.5	**26.3**/84.2	24.8/72.6
CodeQA	P1: Direct	8.7/15.6	20.4/24.3	24.0/52.7	9.2/14.8	22.1/27.6	26.3/58.4	7.4/13.2	18.6/25.8	21.2/20.1
	P2: RLM	18.5/198.4	36.8/168.2	**58.6**/112.5	16.7/215.6	**46.4**/154.3	**62.1**/98.7	12.3/232.8	32.4/182.4	**49.3**/145.6
	P3: λ-RLM	**35.2**/72.3	**47.1**/58.6	54.2/46.8	**32.8**/76.4	43.8/54.2	55.7/42.1	**27.6**/81.5	**42.5**/63.7	44.8/54.3

construction (OOL-Pairs), while open-ended code navigation and adaptive search favor RLM in some frontier cells. Thus restricted control is a structural inductive bias and a partial substitute for scale, not a replacement for model capability. **Ablation Study.** Table 7 shows that the combinator library, plan fixity, and numeric planner all matter. The more diagnostic factorized ablation, exact k-sweep, task-classification sensitivity, measured $\widehat{\mathcal{A}}(\tau)$ and $\widehat{\mathcal{A}}_{\oplus}$, and leaf-correlation study are reported in Appendix D.

Table 4: Summary of Table 3. The "λ-RLM higher" and "RLM higher" counts of mean accuracy sum to 45 (36 vs. 9); five-seed tests classify the 45 cells as 36 significant λ-RLM wins, 8 ties, and 1 significant RLM win. ΔAcc: mean over the nine models of λ-RLM minus RLM accuracy (points). Speedup: mean RLM latency divided by mean λ-RLM latency.

Model Tier Analysis			**Task Complexity Analysis**				
Tier	λ-RLM higher	Rate	Task	ΔAcc	Speedup	RLM higher	Complexity
Weak (8B/7B)	15/15	100%	S-NIAH	+10.1	3.5×	1/9	$O(1)$
Medium	14/15	93%	OOLONG	+14.3	3.7×	2/9	$O(n)$
Strong	7/15	47%	OOL-Pairs	+27.0	6.2×	0/9	$O(n^2)$
All	**36/45**	**80%**	BrowseComp+	+8.0	4.3×	2/9	m-hop
			CodeQA	+5.6	2.7×	4/9	Variable
			All	**+13.0**	**4.0×**	**9/45**	

7 Validating the Analysis and Planner

Are the accuracy assumptions predictive? We measure leaf accuracy against per-chunk generator labels and measure composition reliability by supplying gold leaf outputs to $\oplus$. Table 5 reports three representative tasks. The bounds are below observed end-to-end performance, but the regimes differ: OOLONG and CodeQA use the conjunctive criterion in Eq. equation 8, whereas S-NIAH uses the relevant-leaf criterion in Eq. equation 9. Symbolic composition has $\widehat{\mathcal{A}}_{\oplus} = 1$ by construction; neural composition does not. We find mild positive leaf correlation on OOLONG ($\bar{r} = .11$) and S-NIAH ($\bar{r} = .04$). Consequently, the independence product is conservative in these measurements; negative correlation would reverse that conclusion, so A7c remains an explicit assumption.

Table 5: Measured theoretical quantities on Qwen3-8B. "Bound" uses the task-appropriate case of Theorem 3.

Task	Criterion / $\oplus$	τ^*	L	$\widehat{\mathcal{A}}(\tau^*)$	$\widehat{\mathcal{A}}_{\oplus}$	Bound	Observed
S-NIAH	relevant / FilterBest	32K	4	.70	1.00	.240	.246
OOLONG	conjunctive / Merge	26.2K	5	.85	1.00	.444	.483
CodeQA	conjunctive / $\mathcal{M}\circ$Concat	32K	8	.79	.81	.123	.352

Does the corrected fan-out rule match behavior? For OOLONG 131K with $K = 32$K, one-level choices below $k_0 = \lceil 131/32 \rceil = 5$ are infeasible. Table 6 shows that $k = 5$ is also the best tested admissible value: larger fan-out creates more, shorter leaves and degrades accuracy and latency. This is consistent with Proposition 4; it directly contradicts the withdrawn universal $k = 2$ claim.

Table 6: Fan-out and planner sensitivity on Qwen3-8B × OOLONG 131K.

Setting	k	τ^*	L	Pred. acc.	Acc.	Latency / change
Smallest feasible	5	26.2K	5	.444	48.3	62.4s
Over-fragmented	8	16.4K	8	.188	22.5	86.0s
Severely fragmented	16	8.2K	16	.013	1.4	170.0s
Oracle task type	5	26.2K	5	–	50.1	+1.8 pp
Predicted type (4.3% errors)	5	26.2K	5	–	48.3	baseline
Half threshold	–	13K	–	–	35.6	−12.7 pp

Comparison with generated planning. On S-NIAH with Qwen3-8B, the one-shot PlanGen baseline obtains 6.17% accuracy at 15.67s, versus 3.2/12.3 for Direct, 8.4/164.3 for RLM, and 24.6/45.9 for λ-RLM (accuracy/latency). Generating a plan once is therefore not sufficient in this test; the restricted executor remains important. This comparison is limited to one model–task pair, so broader comparisons with planning and hierarchical-agent systems remain necessary.

Table 7: Factorized ablation on Qwen3-8B × OOLONG (131K tokens).

Arm	Control configuration	Acc (%)	Lat (s)	ΔAcc	Isolated factor
(a)	Closed term; fixed combinators and plan	48.3	62.4	–	Full λ-RLM
(b)	Same combinators; plan re-chosen each turn	40.2	91.3	−8.1	Plan fixity
(c)	Combinators; $\mathcal{M}$ selects $\oplus$ at runtime	44.6	74.8	−3.7	Runtime operator choice
(d)	Free-form code; combinators available as helpers	31.7	198.4	−16.6	Static restriction
(e)	Free-form code; no combinators (RLM)	24.1	241.6	−24.2	Library + fixed plan

8 Discussion

What the formalism contributes. The practical system can be implemented without a surface λ syntax. Its formal value is narrower and more testable: after task selection and numeric planning, Φ is closed, typed, and independent of model output. Closedness makes "the model cannot rewrite control flow" a property of the program object rather than an operational convention. Parametricity then lets one termination and cost argument cover every well-typed substitution of $(k, \tau, \oplus, \pi)$. A conventional higher-order runtime can offer the same guarantees if it preserves these invariants. This clarification separates the contribution from two stronger claims that we do not make: the calculus neither discovers useful plans nor explains the full empirical gain. Table 7 instead attributes the gain to a combination of a restricted operator library and fixing the plan before recursive execution.

Interpreting the theory. The corrected cost analysis changes the planner qualitatively. Total leaf input is n regardless of fan-out, so the old relaxed nk/τ term penalized large k artificially. The exact objective depends on the rounded leaf count, generated outputs, and internal composition. This produces a simple one-level rule when $\lceil n/K \rceil$ fits the composition fan-in, and a finite integer optimization otherwise. The accuracy analysis is likewise conditional rather than universal. Tasks that require every leaf to be correct retain an exponential leaf-failure term. Retrieval tasks instead fail through an incorrect relevant leaf or accumulated distractor false positives.

Interpreting the empirical gains. The strongest results occur when task structure is known and symbolic work replaces neural control: OOL-Pairs classifies items neurally but constructs the quadratic cross-product symbolically. The weakest relative results occur when a strong coding model benefits from adaptive repository traversal, regex construction, or backtracking beyond the current library. This explains why gains are large at weak and medium scale but narrow at the frontier. The result is therefore best viewed as a reliability–expressivity trade-off: static control lowers the coding burden and variance, while dynamic control offers a larger search space.

Next steps. A natural extension is a two-stage verified controller: allow the model to propose a new combinator or plan, then type-check it, prove a decreasing rank, and estimate its cost before execution. This would preserve the audit boundary while expanding expressivity for code and adaptive search.

9 Conclusion

λ-RLM replaces model-authored recursive control with a closed, typed controller and bounded neural leaves. The revised analysis establishes well-founded recursion, a tight fan-out objective rather than a universal $k = 2$ rule, and accuracy claims that distinguish conjunctive from single-relevant-leaf tasks. Across the evaluated settings, restriction improves predictability and usually improves accuracy and latency, especially for smaller models, while frontier and open-ended tasks expose the cost of reduced flexibility. Future work includes a type-checked extension in which the model proposes new combinators or plans before execution.

AI use statement

AI tools assisted with manuscript drafting, language editing. The authors remain responsible for the mathematical claims and reported experimental results.

REFERENCES

Keivan Alizadeh, Parshin Shojaee, Minsik Cho, and Mehrdad Farajtabar. Recursive language models meet uncertainty: The surprising effectiveness of self-reflective program search for long context, 2026. URL https://arxiv.org/abs/2603.15653.

Varun Pratap Bhardwaj. Formal analysis and supply chain security for agentic ai skills. *arXiv preprint arXiv:2603.00195*, 2026.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent llm systems fail?, 2025. URL https://arxiv.org/abs/2503.13657.

Shengkai Chen, Zhiguang Cao, Jianan Zhou, Yaoxin Wu, Senthilnath Jayavelu, Zhuoyi Lin, Xiaoli Li, and Shili Xiang. Dragon: Llm-driven decomposition and reconstruction agents for large-scale combinatorial optimization, 2026. URL https://arxiv.org/abs/2601.06502.

Zac Garby, Andrew D Gordon, and David Sands. The llmbda calculus: Ai agents, conversations, and information flow. *arXiv preprint arXiv:2602.20064*, 2026.

Bowen Jin, Jinsung Yoon, Jiawei Han, and Sercan O. Arik. Long-context llms meet rag: Overcoming challenges for long inputs in rag, 2024. URL https://arxiv.org/abs/2410.05983.

Zhuowan Li, Cheng Li, Mingyang Zhang, Qiaozhu Mei, and Michael Bendersky. Retrieval augmented generation or long-context llms? a comprehensive study and hybrid approach, 2024. URL https://arxiv.org/abs/2407.16833.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023. URL https://arxiv.org/abs/2307.03172.

Xindi Wang, Mahsa Salmani, Parsa Omidi, Xiangyu Ren, Mehdi Rezagholizadeh, and Armaghan Eshaghi. Beyond the limits: A survey of techniques to extend the context length in large language models, 2024. URL https://arxiv.org/abs/2402.02244.

Zhen Xu, Shang Zhu, Jue Wang, Junlin Wang, Ben Athiwaratkun, Chi Wang, James Zou, and Ce Zhang. When does divide and conquer work for long context llm? a noise decomposition framework, 2026. URL https://arxiv.org/abs/2506.16411.

Zhuoyi Yang, Xu Guo, Tong Zhang, Huijuan Xu, and Boyang Li. Test-time scaling of llms: A survey from a subproblem structure perspective, 2025. URL https://arxiv.org/abs/2511.14772.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models, 2026. URL https://arxiv.org/abs/2512.24601.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models, 2023. URL https://arxiv.org/abs/2205.10625.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, Xiaoteng Ma, Xiaodong Yu, Gowtham Ramesh, Jialian Wu, Zicheng Liu, Pan Lu, James Zou, and Jiaxuan You. Where llm agents fail and how they can learn from failures, 2025. URL https://arxiv.org/abs/2509.25370.

A λ-CALCULUS EXAMPLES

A.1 EXAMPLES OF β-REDUCTION

Examples of β-Reduction

Identity. Let $\texttt{id} = \lambda x.\, x$. Then $(\lambda x.\, x)\; y \xrightarrow{\beta} y$: applying the identity returns its input.
Constant function. Let $\texttt{const} = \lambda x.\, \lambda y.\, x$. By left associativity, $\texttt{const}\; a\; b = ((\lambda x.\, \lambda y.\, x)\; a)\; b$. Reducing step by step:

$$\underbrace{(\lambda x.\, \lambda y.\, x)\; a}_{\xrightarrow{\beta}\, \lambda y.\, a}\; b \xrightarrow{\beta} a.$$

A.2 WORKED EXAMPLE: RECURSION WITH A FIXED-POINT OPERATOR

Worked Example: Recursion with a Fixed-Point Operator

Step 1: Write the recipe. $G \triangleq \lambda f.\, \lambda n.\, \textbf{if } (n = 0) \textbf{ then } 1 \textbf{ else } n \cdot f(n-1)$.
Step 2: Tie the recursive definition. Define $\texttt{fact} \triangleq \textbf{fix}(G)$. By the fixed-point property,

$$\texttt{fact} = \textbf{fix}(G) = G(\textbf{fix}(G)) = G(\texttt{fact}),$$

which supplies the recipe with its own recursive function.
Step 3: Trace $\texttt{fact}(3)$. $\texttt{fact}(3) \xrightarrow{\beta} 3 \cdot \texttt{fact}(2) \xrightarrow{\beta} 3 \cdot 2 \cdot \texttt{fact}(1) \xrightarrow{\beta} 3 \cdot 2 \cdot 1 \cdot \texttt{fact}(0) \xrightarrow{\beta} 6$.

Why λ-calculus, not just recursive Python? The implementation is ordinary strict code; β-reduction and recursive calls are operationally equivalent here. The calculus is a specification language for a fixed higher-order controller: closedness states syntactically that model outputs cannot alter control flow, types validate plans offline, and one parameterized executor supports uniform proofs. An equivalently verified higher-order Python function could provide the same guarantees. The contrast is therefore static versus model-generated control, not λ versus Python.

B ADDITIONAL DETAILS ON THE FRAMEWORK

B.1 EXECUTION DIAGRAM

The execution diagram for the λ-RLM, and how it results in β-reduction is shown below:

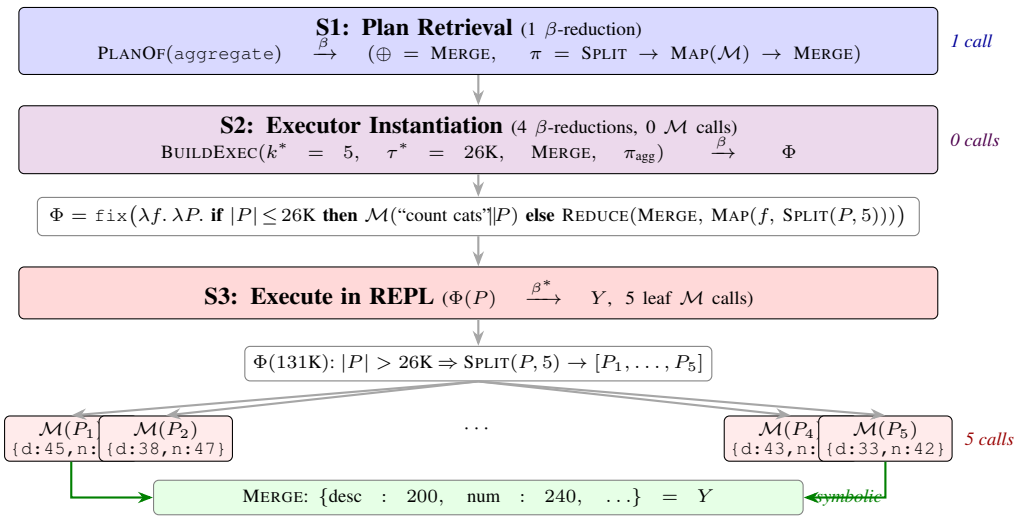
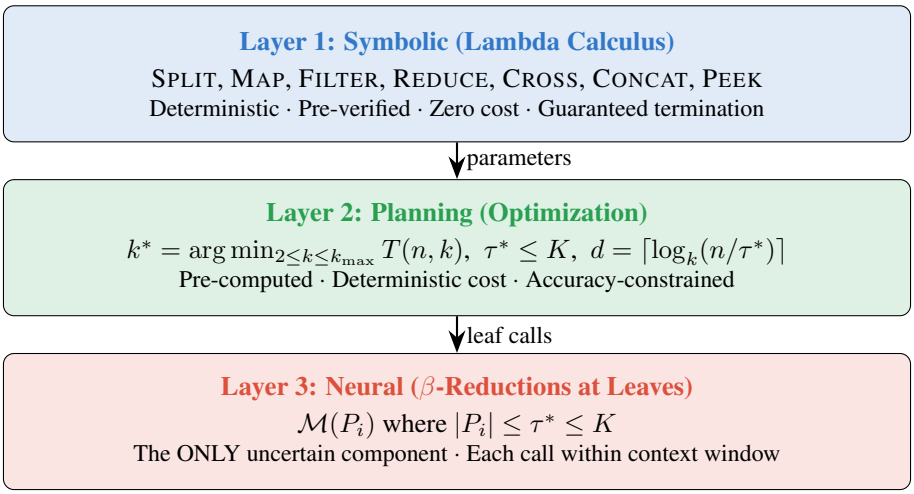


Figure 3: **The three-stage β-reduction pipeline** on OOLONG (131K tokens, $K = 32$K). **S1**: one β-reduction retrieves $(\oplus, \pi)$. **S2**: four substitutions produce a closed executor Φ. **S3**: $\Phi(P)$ unfolds into 5 leaf calls whose results are merged symbolically.

B.2 The Hierarchy of Computation



B.3 What Beta Reduction Gives: A Worked Search Example

We trace through SEARCH concretely to show exactly what λ-calculus contributes and what it does not.

What lambda calculus does NOT do. Given Panel A (the combinator library) and a task type, λ-calculus cannot derive the composition operator $\oplus$ or the execution plan π. Those are *human design decisions*: we looked at each task, chose which combinators make sense, and wrote them into Panel B. Lambda calculus had no role in that step.

What lambda calculus DOES do. Once Panel B exists, λ-calculus encodes all rows into the parameterized term PLANOF. The guarantees apply to the shared executor after substitution. The same organization is possible with a verified higher-order function; the term makes the static interface explicit.

The Python Baseline

Without λ-calculus, the natural implementation is one function per task:

```
def search(P):
    chunks   = split(P, k=3)
    filtered = filter(relevant, chunks)
    results  = [M(c) for c in filtered]
    return best(results)

def aggregate(P):
    chunks  = split(P, k=3)
    results = [M(c) for c in chunks]
    return merge(results)
```

These two ad hoc functions do not expose a shared controller. Refactoring them into one typed higher-order function would recover the same proof structure as BUILDEXEC.

Stage 1: Encoding Panel B as One Lambda Term

All rows of Panel B are encoded into PLANOF. Applying it to a task type is one β-reduction:

$$\text{PLANOF (SEARCH)} \xrightarrow{\beta} \big(\text{FILTERBEST},\ \text{SPLIT} \to \text{MAP(PEEK)} \to \text{FILTER} \to \text{MAP}(\mathcal{M}) \to \text{BEST}\big). \tag{10}$$

This is *retrieval by substitution*, not derivation. The value is that the same mechanism retrieves any plan, so one proof about PLANOF covers all of them.

STAGE 2: EXECUTOR INSTANTIATION

$k^* = 3$, $\tau^* = 50$ from the planner. Substituting into BUILDEXECUTOR:

$$\text{BUILDEXECUTOR}(k^*{=}3,\ \tau^*{=}50,\ \oplus{=}\text{FILTERBEST},\ \pi{=}\pi_{\text{search}}) \xrightarrow{\beta} \Phi \tag{11}$$

Φ is fully closed. It is fixed before any prompt arrives.

STAGE 3: EXECUTION ON A REAL PROMPT

P = `"The quick brown fox jumps over the lazy dog near the river where the old fisherman caught a big salmon on tuesday"`

Step 1 Unroll fix: $|P| = 145 > 50$, else branch fires.

Step 2 SPLIT (symbolic): P_1 = `"The quick brown fox jumps over the lazy dog"`, P_2 = `"near the river where the old fisherman caught"`, P_3 = `"a big salmon on tuesday morning before the storm"`

Steps 3–4 MAP(PEEK) then FILTER (both symbolic, zero LLM cost): P_1 dropped (no fisherman/salmon); $[P_2, P_3]$ retained.

Step 5 Base case fires on each chunk ($|P_2|, |P_3| \leq 50$): $\mathcal{M}(P_2) \to$ `"tuesday morning"`; $\mathcal{M}(P_3) \to$ `"tuesday morning before the storm"`

$\mathcal{M}$ called **exactly twice**. All structural decisions were made by β-reduction before $\mathcal{M}$ was invoked.

Step 6 FILTERBEST (symbolic): $Y =$ `"tuesday morning before the storm"`. □

WHY THIS MATTERS

1. **One termination proof covers all plans.** The plans instantiate the same typed executor; balanced splitting strictly lowers its rank.

2. **Cost is computable before execution.** $N(n) \leq (k^*)^d + 1$ for symbolic $\oplus$, and $(k^*)^d + ((k^*)^d - 1)/(k^* - 1) + 1$ for neural $\oplus$ (Theorem 1), bounds the instantiated controller.

3. **The LLM cannot alter the plan.** Φ is a closed term fixed at S2. $\mathcal{M}$ only sees leaf sub-prompts and, for neural $\oplus$, concatenated child outputs.

4. **Adding a task costs one line.** One new case in PLANOF; all existing proofs remain valid.

Summary. Lambda calculus does not derive Panel B from Panel A. It specifies a closed, typed controller after PLANOF and BUILDEXEC are instantiated. The empirical ablation in Table 7 separately measures the value of combinators and plan fixity.

B.4 ALGORITHM DETAIL

Algorithm 3 λ-RLM: Complete System

Input: Prompt $P \in \Sigma^*$, model $\mathcal{M}$ with window K, accuracy target $\alpha \in (0, 1]$
Output: Response $Y \in \Sigma^*$

```
// == Phase 1: REPL Initialization ==
state ← InitRepl(prompt = P)                    ▷ P stored in environment, not in context window
state ← RegisterLibrary(state, L)               ▷ load pre-verified combinators into REPL
state ← RegisterSubCall(state, sub_M)           ▷ register M as REPL-callable leaf solver
// == Phase 2: Task Detection ==
meta ← E(state, Peek(P, 0, 500); len(P))        ▷ symbolic probe, no M call
t_type ← M("Select from T: " ‖ meta)            ▷ single M call: menu selection
// == Phase 3: Short-circuit ==
if |P| ≤ K then                                 ▷ prompt fits in context window
    return sub_M(P)                             ▷ direct call, no decomposition needed
end if
// == Phase 4: Planning (only reached if |P| > K) ==
(⊕, π) ← PlanOf(t_type)                         ▷ finite map encoded by Table 1
k_max ← ⌊K / n̄_out⌋
k* ← arg min_{2≤k≤k_max} T(|P|, k) using Eq. equation 7
τ* ← min{K, ⌈|P|/(k*)^d⌉}, d ← ⌈log_{k*}(|P|/K)⌉   ▷ deterministic arithmetic; one split when feasible
// == Phase 5: Build and Execute Recursive Executor ==
state ← E(state, Φ = BuildExecutor(k*, τ*, ⊕, π, sub_M))
state ← E(state, result = Φ(P))                 ▷ single execution of Φ in REPL
return state[result]
```

Algorithm 4 Φ: Combinator Executor (registered in REPL, not LLM-generated)

Input: Prompt P (from REPL state), parameters $k^*, \tau^*, \oplus, \pi$ (from planner)
Output: Result string

```
function Φ(P)
    if |P| ≤ τ* then
        q ← Template[t_type].Fmt(P)
        return sub_M(q)
    else
        [P_1, ..., P_{k*}] ← Split(P, k*)
        if π includes Filter then
            previews ← Map(λp. Peek(p, 0, ⌊τ*/10⌋), [P_1, ..., P_{k*}])
            [P_1, ..., P_{k'}] ← Filter(Relevant, Zip([P_{1:k*}], previews))
        end if
        [R_1, ..., R_{k'}] ← Map(λp_i. Φ(p_i), [P_1, ..., P_{k'}])
        return Reduce(⊕, [R_1, ..., R_{k'}])
    end if
end function
```

Algorithm 5 Pairwise Tasks

Input: P, predicate ϕ, $\mathcal{M}$, k^*, τ^*
Output: Pairs $\mathcal{S} \subseteq \mathbb{N} \times \mathbb{N}$

$[P_{1:k^*}] \leftarrow \textsc{Split}(P, k^*)$
labels $\leftarrow \textsc{Map}(\texttt{sub_}\mathcal{M}_{\text{cls}}, P_{1:k^*})$
$L \leftarrow \textsc{Parse}(\textsc{Concat}(\text{labels}))$
$Q \leftarrow \textsc{Filter}(\phi, L.\textsc{Items}())$
$\mathcal{S} \leftarrow \{(i,j) \mid i,j \in Q, i < j\}$
return $\mathcal{S}$

Algorithm 6 Multi-Hop Search

Input: Corpus $[D_1, \ldots, D_m]$, query q, $\mathcal{M}$
Output: Answer Y

prev $\leftarrow \textsc{Map}(\textsc{Peek500}, D_{1:m})$
rel $\leftarrow \textsc{Filter}(\textsc{Match}(q), \textsc{Zip}(D, \text{prev}))$
evi $\leftarrow \textsc{Map}(\texttt{sub_}\mathcal{M}_{\text{ext}}, \text{rel})$
prompt $\leftarrow$ "answer "$\|q\|\textsc{Concat}(\text{evi})$
$Y \leftarrow \texttt{sub_}\mathcal{M}(\text{prompt})$
return Y

C PROOFS

Theorem 1 (Termination and call count). *Under A1, A2, and A6, for every $P \in \Sigma^*$ with $|P| = n \in \mathbb{N}$, $\Phi(P)$ terminates. Its depth is at most d and it makes at most $L \le k^d$ leaf calls. Counting task-type selection, the complete system makes*

$$N(n) \le \begin{cases} k^d + 1 & \textit{symbolic } \oplus, \\ k^d + \dfrac{k^d - 1}{k - 1} + 1 & \textit{neural } \oplus \ (\oplus = \mathcal{M} \circ \textsc{Concat}), \end{cases} \tag{6}$$

where the middle term counts one composition call per internal node, whose input (the concatenated child outputs) is assumed to fit in K, i.e. $k \le k_{\max}$.

Proof. Define $r(P') = \max\{0, \lceil \log_k(|P'|/\tau) \rceil\}$. At rank zero, $|P'| \le \tau \le K$ and the leaf call halts by A1. At positive rank, A6a gives $|P_i| \le \lceil |P'|/k \rceil < |P'|$ and therefore $r(P_i) < r(P')$; A6b can only remove children. Strong induction terminates every recursive call, and A2 terminates the enclosing combinators. A depth-d tree has at most k^d leaves and at most $1 + k + \cdots + k^{d-1} = (k^d - 1)/(k - 1)$ internal nodes. A neural $\oplus = \mathcal{M} \circ \textsc{Concat}$ makes one call at each internal node on at most $k\bar{n}_{\text{out}} \le K$ tokens, so it halts by A1; a symbolic $\oplus$ makes none by A2. Adding the one task-type call gives $N(n) \le k^d + 1$ for symbolic $\oplus$ and $N(n) \le k^d + (k^d - 1)/(k - 1) + 1$ for neural $\oplus$. □

Theorem 2 (Tight cost expression). *For a full balanced tree, per-input-token cost c_{in}, mean leaf output length $\bar{n}_{\text{out}}$, per-output-token cost c_{out}, and linear internal composition cost $c_{\oplus}k$, the exact cost (up to fixed planner overhead) is*

$$T(n, k) = c_{\text{in}} n + c_{\text{out}} \bar{n}_{\text{out}} L + c_{\oplus} k \frac{L - 1}{k - 1}, \qquad L = k^d. \tag{7}$$

Filtering only decreases this value. If composition is symbolic, $c_{\oplus} = 0$; cost then depends on k only through the rounded leaf count L, and can be identical for several fan-outs.

Proof. At recursion level j there are k^j nodes. Hence a full depth-d tree has $L = k^d$ leaves and $1 + k + \cdots + k^{d-1} = (L - 1)/(k - 1)$ internal nodes. Because the leaves partition the original prompt without overlap, their input lengths sum to exactly n, giving input cost $c_{\text{in}} n$ rather than the relaxed surrogate $L\, c_{\text{in}} \tau$. Leaf outputs contribute $c_{\text{out}} \bar{n}_{\text{out}} L$, and each internal node contributes $c_{\oplus} k$. Summing yields Eq. equation 7. Contractive filtering removes subtrees and cannot increase the total. □

Theorem 3 (Task-appropriate accuracy bounds)**.** *Under A4–A7:* (i) *if $n \le \tau$ (equivalently $d = 0$), then $\mathcal{A}_{\lambda\text{-RLM}}(n) = \mathcal{A}(n) \ge \mathcal{A}(\tau)$;* (ii) *for conjunctive tasks,*

$$\mathcal{A}_{\lambda\text{-RLM}}(n) \ge \mathcal{A}(\tau)^L \mathcal{A}_{\oplus}^d; \tag{8}$$

thus the general bound retains an exponential term in $L = \Theta(n/\tau)$ whenever $\mathcal{A}(\tau) < 1$; (iii) *only in the limiting case $\mathcal{A}(\tau) = 1$ does the remaining $\mathcal{A}_{\oplus}^d$ term yield power-law decay in n; and* (iv) *for localisable retrieval,*

$$\mathcal{A}_{\lambda\text{-RLM}}(n) \ge \mathcal{A}(\tau)(1 - p_{\text{fp}})^{L-1} \mathcal{A}_{\oplus}^d. \tag{9}$$

With symbolic composition and $p_{\text{fp}} = 0$, the retrieval bound is constant in n.

Proof. If $n \le \tau$, the executor makes one call and A4 gives case (i). For a conjunctive task, A7c lower-bounds joint leaf success by $\mathcal{A}(\tau)^L$; multiplying the d neural composition factors proves Eq. equation 8. Substituting $\mathcal{A}(\tau) = 1$ before any exponent relaxation proves case (iii). For a localisable task, success requires the relevant leaf to be correct, every one of the other $L - 1$ leaves to avoid a surviving false positive, and every neural composition to preserve the answer. A7r gives Eq. equation 9. Setting $p_{\text{fp}} = 0$ and $\mathcal{A}_{\oplus} = 1$ yields the constant special case. □

Proposition 4 (Fan-out planning)**.** *Let $2 \le k \le k_{\max}$, where $k_{\max}$ enforces composition fan-in. The cost-minimizing fan-out is the finite minimizer of equation 7; $k = 2$ is not generally optimal. If a one-level split is feasible, $k_0 = \lceil n/K \rceil \le k_{\max}$, then $k^* = k_0$, $\tau^* = \lceil n/k^* \rceil$, and $d^* = 1$. Otherwise the planner enumerates the admissible integers using the exact objective.*

Proof. Equation equation 7 is explicit for every integer $2 \le k \le k_{\max}$, so exhaustive evaluation returns the exact finite minimizer. Suppose $k_0 = \lceil n/K \rceil \le k_{\max}$. Then k_0 produces one level and the smallest feasible number of leaves. Any $k > k_0$ also has depth one but strictly more leaves and no fewer internal-node arguments, so nonnegative output and composition costs cannot decrease. Multi-level trees with $k < k_0$ are no better: they need $L = k^d \ge k_0$ leaves and their composition term satisfies $k(L-1)/(k-1) \ge L \ge k_0$ (as $L \ge k$), so their leaf-output and composition costs are at least those of k_0, while the input term $c_{\text{in}} n$ is identical. Thus $k^* = k_0$, $\tau^* = \lceil n/k_0 \rceil \le K$, and $d^* = 1$. When one level is infeasible, rounding in $L = k^{\lceil \log_k(n/K) \rceil}$ prevents a universal closed-form k; finite enumeration is exact. □

D Additional Experimental Results

D.1 Experiment Detail

We evaluate λ-RLM across three open-source model families, each with three strength tiers (Table 8): Qwen3 (8B, 32B, 235B-A22B), Llama (3.1-8B, 3.3-70B, 3.1-405B), and Mistral (7B-v0.3, Mixtral-8x22B, Codestral-22B). All models are served via vLLM on 8×H100 nodes with tensor parallelism. In every configuration, the same model serves as both the root (task detection) and the leaf (sub-call) solver; we do not use a stronger model for orchestration and a weaker one for leaves.

Table 9 lists the five benchmarks used in our evaluation. The tasks span constant, linear, quadratic, multi-hop, and variable complexity: S-NIAH ($O(1)$, single needle retrieval), OOLONG ($O(n)$, per-entry aggregation), OOLONG-Pairs ($O(n^2)$, pairwise comparison), BrowseComp+ ($O(1)$ multi-hop over 6–11M tokens), and CodeQA (variable complexity, repository understanding). Each task is run at context-length buckets {8K, 16K, 32K, 64K, 128K} where applicable; BrowseComp+ uses fixed 6–11M token inputs where P1: Direct scores 0.0^* as the input exceeds all context windows. We report macro-averages across non-empty buckets.

For P1 (Direct), the entire prompt is fed to the model in one call; when the input exceeds K, the run is marked as a failure. P2 uses the original RLM system and prompts of Zhang et al. (2026). P3 chooses fan-out by evaluating Eq. equation 7 over the feasible integers. Headline configurations use five seeds. We compare P2 and P3 with paired two-sided permutation tests at $\alpha = 0.05$; mean differences below the test threshold are ties. We collect task accuracy/F1, wall-clock latency, and model-call count.

Table 8: Model families and strength tiers. Tiers are relative within each family: smallest (weak), middle (medium), and largest (strong) model.

Family	Property	Weak	Medium	Strong
Qwen3	Model	Qwen3-8B	Qwen3-32B	Qwen3-235B-A22B
	Context	32K	128K	128K
Llama	Model	Llama-3.1-8B	Llama-3.3-70B	Llama-3.1-405B
	Context	128K	128K	128K
Mistral	Model	Mistral-7B-v0.3	Mixtral-8x22B	Codestral-22B
	Context	32K	64K	32K

Table 9: Benchmark tasks by complexity.

Task	Complexity	Tokens	Metric	Inst.	λ-RLM plan π
S-NIAH	$O(1)$	8K–128K	F1	100	$\textsc{Split} \to \textsc{Map}(\textsc{Peek}) \to \textsc{Filter} \to \textsc{Map}(\mathcal{M}) \to \textsc{Best}$
OOLONG	$O(n)$	8K–128K	Score	50	$\textsc{Split} \to \textsc{Map}(\mathcal{M}) \to \textsc{Merge}$
OOL-Pairs[2]	$O(n^2)$	8K–128K	F1	20	$\textsc{Split} \to \textsc{Map}(\mathcal{M}) \to \textsc{Parse} \to \textsc{Filter} \to \textsc{Cross}$
BrowseComp+	$O(1)$ m-hop	6M–11M	Acc	200	$\textsc{Split}_\delta \to \textsc{Map}(\textsc{Peek}) \to \textsc{Filter} \to \textsc{Map}(\mathcal{M}) \to \mathcal{M}$
CodeQA	Variable	23K–4.2M	Acc	23	$\textsc{Split} \to \textsc{Map}(\mathcal{M}) \to \textsc{Concat} \to \mathcal{M}$

D.2 Rebuttal Experiments and Assumption Checks

Measured leaf and composition reliability. Leaf accuracy is measured against per-chunk labels from each benchmark generator. Composition reliability is measured by feeding gold leaf outputs to $\oplus$, isolating composition failure. Table 10 shows that the analytical quantities are measurable. $\widehat{\mathcal{A}}_\oplus = 1$ for symbolic operators by construction; it is below one for neural composition.

Table 10: Measured terms in Theorem 3 (Qwen3-8B).

Task	Criterion / $\oplus$	τ^*	L	$\widehat{\mathcal{A}}(\tau^*)$	$\widehat{\mathcal{A}}_\oplus$	Bound	Observed
S-NIAH	relevant leaf / FilterBest	32K	4	.70	1.00	.240	.246
OOLONG	conjunctive / Merge	26.2K	5	.85	1.00	.444	.483
CodeQA	conjunctive / $\mathcal{M}\circ$Concat	32K	8	.79	.81	.123	.352

Leaf dependence. Table 11 measures mean pairwise outcome correlation over five seeds. Correlations are mildly positive, so the independence product is conservative on these tasks. A correlation-adjusted estimate slightly overpredicts; the observed joint rate lies between the two estimates. Negative correlation would instead make A7c optimistic.

Exact fan-out sweep. For OOLONG 131K with $K = 32$K, $k < 5$ is infeasible because a one-level leaf would exceed the window. Table 12 verifies that the smallest feasible one-level fan-out predicted by Proposition 4 is best among the tested admissible values. Measured $\widehat{\mathcal{A}}(\tau^*)$ falls as leaves shrink (.85, .81, .76), so A4 does not hold on this task, where shorter leaves split records across boundaries. The bounds in Table 10 therefore use $\widehat{\mathcal{A}}$ measured at the leaf size actually run, and A4 is only a worst-case simplification.

Planner sensitivity and engineering. Task selection is one LLM menu choice over a 500-token probe; the six $(\oplus, \pi)$ rows are human-designed once, while (k, τ, d) are deterministic arithmetic. Adding a seventh task family requires one typed row but no new generic termination or cost proof. Table 13 shows that numeric over-fragmentation matters more than the observed 4.3% task-type error rate.

[2]See Appendix D.1 of Zhang et al. (2026) for the OOLONG-Pairs benchmark.

Table 11: Leaf-outcome dependence. "Adjusted" is an estimate, not a certified bound.

Task	Mean pairwise $\bar{r}$	Independence product	Observed joint	Adjusted estimate
OOLONG 131K	.11	.444	.483	.530
S-NIAH 128K	.04	.240	.246	.265

Table 12: Fan-out sweep on Qwen3-8B $\times$ OOLONG 131K.

k	τ^*	d	L	$\widehat{\mathcal{A}}(\tau^*)$	Predicted acc.	Observed acc. / latency
< 5	> 32K	–	–	–	–	infeasible
5	26.2K	1	5	.85	.444	48.3 / 62.4
8	16.4K	1	8	.81	.188	22.5 / 86.0
16	8.2K	1	16	.76	.013	1.4 / 170.0

Structured-planning baseline. On S-NIAH with Qwen3-8B, PlanGen reaches 6.17% accuracy at 15.67s, compared with 3.2/12.3 for Direct, 8.4/164.3 for RLM, and 24.6/45.9 for λ-RLM (accuracy/latency). This limited comparison suggests that producing a plan once is insufficient without a reliable restricted executor; broader decomposition-baseline evaluation remains future work.

D.3 Comparison with Other Long-Context Paradigms

Table 14 reproduces, without re-running, the baseline results of Zhang et al. (2026): accuracy, with the average API cost in USD and its standard deviation in gray. Those experiments use GPT-5 (RLM sub-calls to GPT-5-mini), Qwen3-Coder-480B-A35B, and Claude Opus 4.1, none of which we ran; our λ-RLM and RLM rows use open-source models accessed through free API services, so no monetary cost was incurred or measured; we report latency instead (Table 3) and these rows carry no cost figures. All numbers should therefore be read across backbones, not as a controlled comparison.

D.4 When Standard RLM Still Wins

Table 15 identifies the nine cells where RLM has a higher mean. They cluster around frontier-scale adaptive search and code navigation, but only one difference is significant at $\alpha = 0.05$. The pattern nevertheless illustrates the expressivity trade-off: fixed combinators improve regularity and efficiency but exclude some instance-specific parsing, backtracking, and search heuristics.

D.5 Targeted Capability Comparisons

Table 16 highlights system-level comparisons. Qwen3-8B improves by 19.7 points over its RLM counterpart while running $4.1\times$ faster. Against Llama-70B RLM, it recovers 93% of the accuracy (31.1 vs. 33.4) at $3.2\times$ lower latency; we do not call this a tie. An 8B λ-RLM also exceeds Direct Llama-405B in mean accuracy, although direct inference is faster. These results support a partial scale-substitution claim only in regimes where systematic decomposition dominates open-ended exploration.

In this table,(ref Table 16) we have shown targeted hypothesis comparisons

Table 16 presents targeted comparisons that directly test our hypotheses.

Table 13: Planner sensitivity on Qwen3-8B × OOLONG 131K.

Perturbation	Accuracy	Change from 48.3
Oracle task type	50.1	+1.8 pp
Predicted task type (4.3% errors)	48.3	–
$k^* \times 1.5$ ($k = 8$)	22.5	−25.8 pp
$\tau^* \times 0.5$ (13K)	35.6	−12.7 pp

Table 14: Full comparison. Rows from Zhang et al. (2026) are copied from their main results table. *: ran into input context limits. Bold: best per column within a backbone block.

Method	**CodeQA**	**BrowseComp+ (1K)**	**OOLONG**	**OOLONG-Pairs**
Task length N (tokens)	23K–4.2M	6M–11M	131K	32K
GPT-5 (RLM sub-calls to GPT-5-mini)				
Base model	24.0* ($0.13±0.07)	0.0*	44.0 ($0.14±0.02)	0.1 ($0.16±0.10)
CodeAct (+ BM25)	22.0* ($0.06±0.08)	51.0 ($0.71±1.20)	38.0 ($0.61±1.06)	24.7 ($0.75±0.43)
CodeAct (+ sub-calls)	24.0* ($0.06±0.08)	0.0*	40.0 ($0.85±1.27)	28.4 ($1.11±0.62)
Compaction agent	58.0 ($1.31±1.46)	70.5 ($0.57±0.10)	46.0 ($0.13±0.01)	0.1 ($0.13±0.09)
OpenCode	18.0*	0.0*	32.0	3.1
OpenCode (+ context offloading)	64.0	**94.0**	52.0	4.8
RLM (depth = 0)	58.0 ($0.18±0.56)	88.0 ($0.44±0.90)	36.0 ($0.37±0.42)	43.9 ($0.69±1.16)
RLM (depth = 1)	62.0 ($0.11±0.10)	91.3 ($0.99±1.22)	56.0 ($0.43±0.85)	58.0 ($0.33±0.20)
RLM (depth = 2)	**66.0** ($0.15±0.30)	92.0 ($0.55±0.69)	56.5 ($1.10±3.25)	65.5 ($0.33±0.44)
RLM (depth = 3)	58.0 ($0.15±0.27)	92.0 ($0.51±0.54)	**58.0** ($0.51±0.54)	**76.0** ($0.39±0.32)
Qwen3-Coder-480B-A35B				
Base model	20.0* ($0.13±0.08)	0.0*	36.0 ($0.06±0.00)	0.1 ($0.05±0.01)
CodeAct (+ BM25)	24.0* ($0.17±0.08)	12.7 ($0.39±0.50)	38.0 ($1.51±1.09)	0.3 ($1.54±0.35)
CodeAct (+ sub-calls)	26.0* ($0.28±0.30)	0.0*	32.0 ($1.83±1.14)	0.1 ($1.49±0.46)
Compaction agent	50.0 ($1.26±1.50)	38.0 ($8.98±2.12)	44.1 ($0.15±0.01)	0.31 ($0.05±0.00)
OpenCode	12.0*	0.0*	36.0	0.0
OpenCode (+ context offloading)	40.0	58.0	24.0	2.1
RLM (depth = 0)	**66.0** ($0.18±0.58)	46.0 ($0.82±0.69)	43.5 ($0.32±0.13)	17.3 ($1.77±1.23)
RLM (depth = 1)	56.0 ($0.92±1.23)	44.7 ($0.84±0.63)	**48.0** ($0.61±0.49)	**23.1** ($1.02±0.52)
RLM (depth = 2)	54.0 ($1.88±3.30)	68.0 ($1.05±0.67)	26.0 ($1.03±1.65)	19.0 ($1.61±0.99)
RLM (depth = 3)	44.0 ($1.65±1.63)	**68.7** ($1.10±0.80)	32.0 ($0.80±1.03)	21.1 ($1.67±1.21)
Claude Opus 4.1				
Claude Code	12.0* ($2.03±0.57)	0.0*	40.2 ($3.43±1.60)	0.1 ($6.75±3.57)
Claude Code (+ context offloading)	62.0 ($1.25±0.54)	84.0 ($2.03±1.49)	48.0 ($0.98±0.55)	6.5 ($2.99±1.16)
Ours (five-seed means; open-source models via free APIs, so no cost)				
RLM, Qwen3-235B	58.6	32.8	56.5	38.4
λ-RLM, Qwen3-235B	54.2	42.1	68.4	61.5
RLM, Llama-405B	62.1	38.4	63.8	42.7
λ-RLM, Llama-405B	55.7	36.2	61.7	64.3

Table 15: The 9 cells where RLM has a higher mean. Eight are statistical ties; Codestral-22B on BrowseComp+ is the significant RLM win.

Task	**Model**	**Tier**	**RLM**	**λ-RLM**	**Why RLM wins**
S-NIAH	Llama-405B	Strong	**52.4**	49.8	Creative regex search
OOLONG	Llama-405B	Strong	**63.8**	61.7	Adaptive batch sizing
OOLONG	Codestral-22B	Strong	**48.2**	45.6	Optimal iteration code
BrowseComp+	Llama-405B	Strong	**38.4**	36.2	Multi-doc search strategies
BrowseComp+	Codestral-22B	Strong	**28.5**	24.8	Code-based document parsing
CodeQA	Qwen3-235B	Strong	**58.6**	54.2	Free-form repo navigation
CodeQA	Llama-70B	Medium	**46.4**	43.8	File-level decomposition
CodeQA	Llama-405B	Strong	**62.1**	55.7	Multi-pass backtracking
CodeQA	Codestral-22B	Strong	**49.3**	44.8	Code-aware chunking

Table 16: Targeted comparisons. All values are averages across 5 tasks.

#	Comparison	Acc (%)	Lat (s)	Verdict
C1	λ-RLM (8B) vs RLM (8B)	31.1 vs 11.4	65 vs 269	**+19.7 pp, 4.1$\times$ faster**
C2	λ-RLM (8B) vs RLM (70B)	31.1 vs 33.4	65 vs 207	**93% accuracy, 3.2$\times$ faster**
C3	λ-RLM (8B) vs Direct (405B)	31.1 vs 21.8	65 vs 39	**8B+λ beats 405B accuracy**
C4	λ-RLM (7B) vs λ-RLM (Cdsrl)	25.1 vs 40.6	74 vs 50	Coding helps, gap = 15.5 pp
C5	RLM (405B) vs λ-RLM (405B)	51.9 vs 53.5	133 vs 39	**λ-RLM wins avg; RLM wins 4/5**