RETREVAL: REASONING TREE WITH VALIDATION AND CROSS-PROBLEM MEMORY FOR LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Every existing inference-time reasoning framework discards failure context at problem boundaries, leaving a model solving problem 500 no wiser than it was on problem 1. We present **ReTreVal** (Reasoning Tree with Validation), a training-free framework combining adaptive tree exploration, tool-augmented refinement, typed-failure backtracking, dual-critique scoring, and persistent self-rewriting memory. On the full MATH-500 benchmark, ReTreVal achieves **85.8%** pass@1 with ground-truth-assisted verification and **84.2%** (421/500) without reference answers in the inference loop. Under label-free verification on MMLU-Pro, it reaches **54.4%** (544/1000), compared with 39.1% for Self-Refine. On full-scale label-free MATH-500, cross-problem Reflexion and Self-Refine with persistent memory reach 81.0% and 83.2%, respectively. Across three full-scale reshuffles, MATH-500 varies from 84.8% to 86.2% (1.4 points) and MMLU-Pro from 53.6% to 54.9% (1.3 points), providing no evidence of a large sequence effect in the tested runs. These results position ReTreVal as an auditable structured-inference procedure that performs targeted recovery and retains bounded failure experience without modifying model weights.

1 INTRODUCTION

When a model fails to solve a problem, the failure is informative: it reveals which approach was tried, where reasoning broke down, and what class of error was made. Yet every existing inference-time reasoning framework discards this information the moment a problem ends. A model attempting problem 500 of a benchmark begins from precisely the same uninformed state as it did on problem 1, despite having encountered, and potentially recovered from, hundreds of structurally similar failures in between. This raises a concrete research question: *can a fixed, unmodified LLM accumulate typed failure experience across a problem sequence and act on it, without any parameter update?*

Prior work, including ReAct (Yao et al., 2023b), Reflexion (Shinn et al., 2023), Self-Refine (Madaan et al., 2023), Tree-of-Thoughts (Yao et al., 2023a), and LATS (Zhou et al., 2023), addresses individual failure modes *within* a single problem but discards all failure context at problem boundaries and retries blindly without categorising why a branch failed. The gap is architectural: no existing framework provides the typed failure accumulation, cross-problem memory, or inference-time self-improvement needed to answer the question above.

We introduce **ReTreVal** (Reasoning Tree with Validation), a training-free framework that answers this question affirmatively. Five tightly coupled mechanisms (adaptive tree construction, adaptive tool-augmented refinement, typed-failure backtracking, critique-driven dual scoring with skepticism penalties, and self-rewriting memory) are described in detail in Section 4. Together they enable inference-time cross-problem learning: a model running ReTreVal on problem 500 has genuinely refined its strategies based on what failed on problems 1–499, without any weight update.

We validate on MATH-500 (Hendrycks et al., 2021) and MMLU-Pro (Wang et al., 2024). On full MATH-500, ReTreVal achieves **85.8%** with ground-truth-assisted verification and **84.2%** under label-free verification; on label-free MMLU-Pro it achieves **54.4%**. Full-scale order reshuffles produce narrow observed ranges of 84.8–86.2% on MATH-500 and 53.6–54.9% on MMLU-Pro.

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.

2 CONTRIBUTIONS

- **Inference-time cross-problem learning.** The first demonstration that a fixed, unmodified LLM can improve its reasoning accuracy across a problem sequence purely at inference time, via self-rewriting memory with typed failure accumulation, LLM-driven entry revision, and confidence-weighted retrieval, with no fine-tuning or reward model.
- **Typed-failure backtracking.** Failures are categorised by root cause and approach type, then injected into the sibling branch's context, structurally preventing repetition of the same error class.
- **Empirical validation.** Consistent gains over all baselines on MATH-500 and MMLU-Pro, with a 3.4:1 win-to-regression ratio confirming genuine error recovery.

3 RELATED WORK

Iterative Reasoning. ReAct (Yao et al., 2023b) interleaves thought-action-observation cycles along a single linear path. Reflexion (Shinn et al., 2023) adds verbal self-reflection after failed attempts, stored in episode-scoped memory, but lacks structured exploration. Self-Refine (Madaan et al., 2023) iteratively critiques and rewrites a single solution. None of these retains memory across problems. LLM-as-judge paradigms (Zheng et al., 2023) and self-consistency (Wang et al., 2023b) improve answer selection but require trained critics or large sample budgets.

Tree Search Methods. Tree-of-Thoughts (ToT) (Yao et al., 2023a) selectively expands promising candidate thoughts; LATS (Zhou et al., 2023), RAP (Hao et al., 2023), and ReAcTree (Choi et al., 2026) add self-reflection, Monte Carlo planning, or hierarchical subgoal expansion. None integrates persistent memory or continuous tool-augmented refinement at each node.

Memory and Learning. Voyager (Wang et al., 2023a) builds a persistent library of executable skills validated in an embodied environment. A static benchmark such as MATH-500 provides neither an execution environment nor an environment-derived success signal, so rather than introduce an artificial proxy, we evaluate Voyager's relevant mechanism—persistent, retrievable cross-problem experience—through cross-problem Reflexion and memory-augmented Self-Refine (Table 2). Generative Agents (Park et al., 2023), MemoryBank (Zhong et al., 2024), and MemGPT (Packer et al., 2023) maintain cross-episode memory but do not target static reasoning benchmarks. TextGrad (Yüksekgönül et al., 2024) optimizes a fixed pipeline's prompts via textual gradients; ReTreVal instead evolves a knowledge base across problems, entirely at inference time.

Positioning. Two capabilities in ReTreVal have no direct precedent. **Typed-failure backtracking**: failures are categorised by root cause and approach type, then injected into the sibling branch so the same error class cannot recur; prior tree-search methods retry with generic reflection. **Inference-time cross-problem learning**: the self-rewriting memory accumulates experience signals, rewrites entries after repeated failures, and tracks per-approach-type reliability, enabling a fixed model to improve its strategy across a benchmark run without parameter updates.

4 RETREVAL ARCHITECTURE AND METHODOLOGY

4.1 SYSTEM OVERVIEW

ReTreVal is orchestrated as a LangGraph (LangChain AI, 2024) state graph (Figure 1) with a fixed pipeline: PLAN → EXPAND → REFINE → CRITIQUE → SCORE → MEMORY → (iterate or synthesize) → VALIDATE → FEEDBACK, with conditional edges for backtracking and decomposition. The components are deliberately coupled: backtracking failure context flows into sibling critique histories, memory insights are injected into every Expand prompt, and the scoring skepticism penalty uses the memory's per-approach-type reliability.

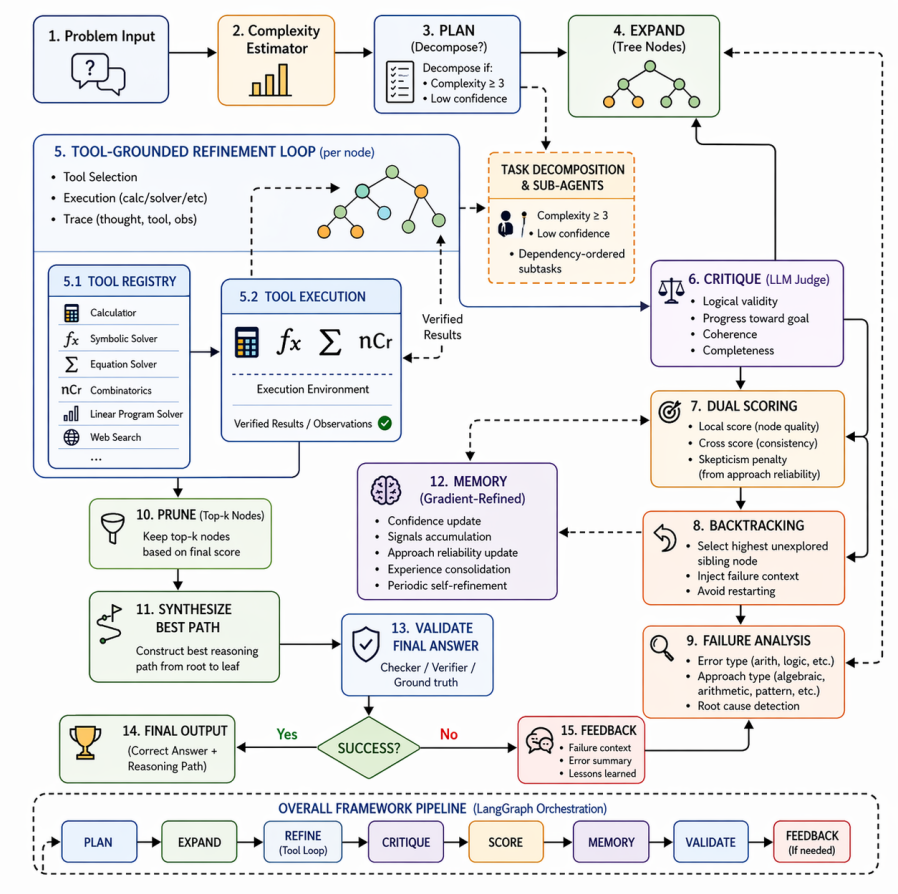


Figure 1: ReTreVal architecture: adaptive tree reasoning with per-node tool-augmented refinement, dual-critique scoring with skepticism penalties, backtracking with failure analysis, self-rewriting memory, and task decomposition for high-complexity problems. Solid arrows: main pipeline; dashed arrows: backtracking recovery on validation failure.

4.2 Adaptive Tree Construction

Fixed-depth tree methods (e.g., ToT at depth 2) either under-explore hard problems or waste compute on easy ones. ReTreVal instead has the LLM rate problem complexity on a 1–5 scale in a single planning call *before* tree construction, which sets depth (2–5) and branching factor (2–4) (Table 6). At each iteration, leaves below max depth generate 2–4 children guided by memory insights and prior failure patterns; after scoring, only the top-k frontier nodes are retained, $\mathcal{L}_{t+1} = \text{top}_k\big(\{n \in \mathcal{L}_t : \text{depth}(n) < d_{\max}\}, \text{score}(n)\big)$, concentrating refinement compute on the most promising paths.

4.3 Adaptive Tool-Augmented Refinement

Pure language model reasoning cannot verify intermediate computations: an incorrect mid-chain result is indistinguishable from a correct one, and later steps silently build on it. At each node, ReTreVal runs an adaptive tool loop (Algorithm 1): the LLM selects the tool suited to the current sub-problem (symbolic equation solver, arithmetic evaluator, combinatorics engine, or LP solver), observes the verified result, and continues until it emits `FINISH`. Three guards prevent runaway execution: exact-duplicate call detection, a consecutive-error threshold of 3, and a ceiling of $S + 2$ LLM calls. Like Toolformer (Schick et al., 2023) and PAL (Gao et al., 2023), this offloads computation to tools, but adaptively per node inside a tree rather than along a linear chain. The registry is extensible: domain-specific tools register as plain callables without framework changes, and tools the LLM does not invoke incur no cost, so capability scales with the tool set without retraining.

4.4 CRITIQUE, SCORING, AND SKEPTICISM

Each node receives a local score s_{local} (self-evaluation against its full solution trace) and a cross score s_{cross} (pairwise critique against its k siblings), combined as score $= 0.6\, s_{\text{local}} + 0.4\, s_{\text{cross}}$. The weighting favours self-evaluation, which sees the full reasoning trace, while incorporating peer feedback from different approaches; flat averaging would dilute the signal when sibling approaches are incompatible. A skepticism penalty then adjusts for historical approach reliability:

$$\text{score}_{\text{final}} = \text{score} \times \big(1 - \alpha(1 - r_a)\big)$$

where r_a is the memory's success ratio for approach type a across all prior problems in the run, updated after every problem, discouraging approach families that have consistently underperformed. The loop terminates when scores stagnate, reach 0.95, or the iteration bound is exceeded.

4.5 SELF-REWRITING MEMORY

Reflexion (Shinn et al., 2023) keeps verbal reflections scoped to a single episode. ReTreVal's memory instead persists *across* problems and *rewrites its own entries* rather than merely appending. Each entry tracks content, category (insight/failure/success), confidence, hit/miss counts, and experience signals; a failure appends its context as a signal. The key novelty is self-rewriting: once an entry accumulates three signals, the LLM rewrites its content to incorporate them, an inference-time self-improvement step on natural language rather than weights that lets the system *revise* a strategy, not just record that it failed. Confidence is updated after each problem as

$$c_{t+1} = \min\!\Big(1,\, c_t + \tfrac{\delta_s}{1+h_t}\Big) \text{ (success)}, \qquad c_{t+1} = \max\!\Big(0,\, c_t - \tfrac{\delta_f}{1+h_t}\Big) \text{ (failure)},$$

where h_t is the hit count and $\delta_s, \delta_f > 0$ are fixed step sizes; the $1 + h_t$ denominator dampens updates for well-tested entries so a single late failure cannot erase accumulated evidence. Entries with confidence < 0.2 are retired every five memory updates, and near-duplicates are merged when the Jaccard similarity of their word sets, $J(e_i, e_j) = |W_i \cap W_j|/|W_i \cup W_j|$, is at least $\theta_{\text{merge}} = 0.6$. Retrieval ranks entries by $\rho(e) = c_e \cdot \max(h_e, 1)$, rewarding both confidence and usage, and injects a confidence-tagged summary of the top entries into every Expand prompt. The memory state serializes to disk, enabling cross-session accumulation.

Memory growth and prompt budget. Retirement and deduplication keep the store from growing append-only, and query-conditioned retrieval injects at most three entries. In the full 500-problem label-free run, the store stabilized at four entries and retrieved context averaged approximately 66 tokens per prompt, so the per-prompt memory contribution is capped and does not grow with the problem index.

4.6 BACKTRACKING AND FAILURE ANALYSIS

When validation rejects an answer, a naive retry, as in Reflexion (Shinn et al., 2023), re-explores the same path with generic feedback. ReTreVal differs in two respects. First, it moves to the highest-scoring unexplored root-level sibling, $n_{\text{next}} = \arg\max_{s \in \mathcal{S} \setminus E} s.\text{combined_score}$, where $\mathcal{S}$ is the set of root children and E the already-explored siblings; since each root child is a distinct high-level approach, recovery commits to a genuinely different strategy rather than a local perturbation. Second, it injects a *typed* failure record, with root cause (arithmetic error, logic error, missing step, wrong constraint, off-by-one) and approach type (algebraic, arithmetic, pattern-matching, brute-force), into the sibling's critique history (Algorithm 2), structurally preventing the same error class from recurring. Backtracking is bounded to one retry to control cost; the 40.9% recovery rate on MATH-500 (§6.5) confirms this budget is sufficient for the majority of recoverable failures.

4.7 TASK DECOMPOSITION

Triggered when complexity ≥ 3 and the best score is < 0.9 after all tree iterations, the decomposer produces 2–4 dependency-ordered subtasks, each solved by a focused sub-agent and aggregated by a final LLM call. Each subtask carries a DONE-WHEN condition that lets its sub-agent self-terminate and provides a checkpoint for backtracking at sub-goal boundaries.

5 EXPERIMENTAL SETUP

5.1 SETUP

Benchmarks. MATH-500 (Hendrycks et al., 2021): 500 problems across 7 subjects and 5 difficulty levels. **MMLU-Pro** (Wang et al., 2024): 1,000 problems sampled from the 10-choice, 14-subject benchmark.

Baselines. Zero-Shot CoT (Kojima et al., 2022; Wei et al., 2022) (single-pass), ReAct (Yao et al., 2023b) (linear tool-augmented chain, 8 steps), Self-Refine (Madaan et al., 2023) (3-cycle single-path refinement), ToT (Yao et al., 2023a) (BFS, depth 2, $k = 3$), LATS (Zhou et al., 2023) (MCTS, 4 iterations). All baselines are reimplemented under identical conditions (same backbone model, same prompt template, same sampling parameters: temp 0.7, top-p 0.95) to isolate the effect of the reasoning framework. Absolute numbers therefore differ from published results, which use method-specific prompt engineering and in some cases different backbone models. The relevant quantity is the margin between ReTreVal and each baseline under the same controlled conditions. For the full-scale label-free MATH-500 comparison, Reflexion and the memory-augmented Self-Refine baseline receive the same persistent cross-problem experience setting as ReTreVal.

Verification protocols. In the original full MATH-500 comparison, reference answers are available to the evaluation pipeline for every method. For methods with iterative control flow, reference-answer correctness determines whether an attempted solution is accepted and whether refinement, backtracking, or memory updates are triggered; for single-pass methods, it is used only to score the completed response. We additionally evaluate ReTreVal and the experience-based comparison methods on all 500 MATH problems under a label-free protocol, in which iterative decisions use only self-critique and tool/answer-consistency signals and reference answers are used only for post-hoc accuracy measurement. MMLU-Pro is label-free throughout and never uses reference answers during inference.

Models and hardware. MATH-500: Qwen 2.5 32B Instruct (Qwen Team, 2024) via vLLM (Kwon et al., 2023) with prefix caching, NVIDIA A100 GPU. MMLU-Pro: Gemini 2.5 Flash (Google DeepMind, 2025) (thinking disabled). Metric: pass@1 accuracy (exact match) for MATH-500 and MMLU-Pro.

Tool Registry. ReTreVal dispatches to a benchmark-specific tool registry at inference time; full details are given in Appendix B.

6 RESULTS AND ANALYSIS

We evaluate ReTreVal against five baselines on MATH-500 and MMLU-Pro. MATH-500 results use Qwen 2.5 32B Instruct; MMLU-Pro results use Gemini 2.5 Flash (thinking disabled). Results are reported as pass@1 accuracy.

6.1 MATH-500 RESULTS WITH GROUND-TRUTH-ASSISTED VERIFICATION

A clear pattern emerges across difficulty levels: ReTreVal's margin over the next-best baseline grows with difficulty, from a tie at Level 1 to +4.5 at Level 2, +7.6 at Level 3, +5.5 at Level 4, and +5.0 at Level 5 (the latter two over Self-Refine). This confirms that tree exploration, backtracking, and critique mechanisms provide the greatest benefit precisely where single-pass methods struggle most. Notably, ReTreVal achieves its highest accuracy at Level 3 (96.2%), surpassing even its Level 1 performance and exceeding the best baseline at this level (LATS, 88.6%) by 7.6 points. Medium-difficulty problems appear to benefit most from structured multi-strategy exploration: they are complex enough to require deliberate reasoning but structured enough for the critique mechanism to reliably identify correct solutions. At the hardest level, all baselines degrade sharply: ReAct drops to 47.0% and even Self-Refine falls to 64.9%, while ReTreVal maintains 69.9%, demonstrating that the combination of tool-augmented reasoning and failure-aware backtracking provides meaningful error recovery where other approaches produce cascading failures.

Table 1 reports accuracy stratified by MATH difficulty level (Level 1 easiest, Level 5 competition-level).

Table 1: Level-wise accuracy (%) on the full MATH-500 benchmark. ReTreVal uses ground-truth-assisted verification in this experiment: reference-answer correctness determines whether backtracking and memory updates are triggered. The **#** column gives the number of problems at each level.

Level	#	ZS-CoT	ReAct	Self-Refine	ToT	LATS	ReTreVal
1	43	93.0	95.3	88.1	90.7	93.0	**95.3**
2	90	88.9	91.1	86.7	86.7	88.9	**95.6**
3	105	86.7	83.8	85.7	84.8	88.6	**96.2**
4	128	73.4	71.9	78.1	74.2	73.4	**83.6**
5	134	60.4	47.0	64.9	64.2	60.4	**69.9**
All	500	77.2	73.2	78.6	77.4	77.6	**85.8**

Table 2: Full-scale label-free comparison on all 500 MATH-500 problems. Reference answers are used only for post-hoc accuracy measurement.

Method	Accuracy
Self-Refine (no memory)	78.6% (393/500)
Reflexion (cross-problem)	81.0% (405/500)
Self-Refine + persistent memory buffer	83.2% (416/500)
ReTreVal (label-free)	**84.2% (421/500)**

6.2 Full-Scale Label-Free MATH-500 Evaluation

We remove reference answers from the inference loop and instead trigger acceptance, backtracking, and memory updates using only self-critique and tool/answer-consistency signals. On the full 500-problem benchmark, ReTreVal achieves 84.2% accuracy (421/500), only 1.6 points below the ground-truth-assisted result (85.8%, 429/500). Cross-problem memory stores only self-derived strategy text and failure patterns, never reference answers or information about future problems. This is an online test-time adaptation setting: each problem is solved before its reference answer is observed, and no future-problem information enters memory. Table 2 compares methods with equivalent access to cross-problem experience.

ReTreVal is highest, but its 1.0-point margin over memory-augmented Self-Refine corresponds to five problems out of 500. The full-scale comparison therefore shows that ReTreVal remains competitive without privileged verification, while the small separation among experience-based methods cautions against attributing its performance to persistent memory alone.

LATS is included in the full MATH-500 comparison as a within-problem tree-search baseline, but it does not retain experience across problems and therefore is not an experience-learning control.

6.3 Compute-Matched Comparison

ReTreVal spends more inference compute than single-pass methods, so its advantage could come from sampling more candidates rather than from its search, backtracking, and memory. We therefore compare it with self-consistency (maj@k) (Wang et al., 2023b), the standard best-of-N baseline, on all 500 MATH-500 problems with the same Qwen2.5-32B-Instruct backbone. Both are label-free: self-consistency commits to the majority answer over k independent samples, ReTreVal to its top-scored attempt, and reference answers are used only for post-hoc scoring. ReTreVal commits after an average of 4.2 attempts per problem, between maj@3 and maj@5, although each attempt also includes tree expansion, critique, and tool calls and is costlier than one sample (Table 3). With fewer candidates than maj@5, ReTreVal's committed accuracy (84.2%, 421/500) exceeds every self-consistency setting: 1.8 pp (nine problems) above maj@8 (82.4%) and 5.2 pp above maj@5 (79.0%). It even exceeds the pass@8 oracle ceiling (83.2%), and its own attempts reach a higher ceiling (pass@5 of 88.6%, 443/500), indicating that its search produces correct candidates that independent sampling does not. Its gains are therefore not explained by additional sampling alone.

Table 3: Compute-matched comparison on all 500 MATH-500 problems (Qwen2.5-32B-Instruct, label-free). Committed accuracy scores the single answer each method commits to (majority vote over k samples for maj@k; top-scored attempt for ReTreVal). pass@k counts a problem as solved if any of the first k samples or attempts is correct; it requires the reference answer and is therefore an oracle ceiling, not an achievable method.

Method	**Samples / problem**	**Committed acc.**	**pass@k (ceiling)**
Self-consistency, maj@1	1	78.2% (391/500)	78.2% (391/500)
Self-consistency, maj@3	3	79.8% (399/500)	80.2% (401/500)
Self-consistency, maj@5	5	79.0% (395/500)	81.6% (408/500)
Self-consistency, maj@8	8	82.4% (412/500)	83.2% (416/500)
ReTreVal (label-free)	4.2 (avg.)	**84.2%** (421/500)	88.6% (443/500, pass@5)

Table 4: Label-free accuracy on 1,000 MMLU-Pro problems using Gemini 2.5 Flash. No method has access to reference answers during inference. ReTreVal improves over the strongest baseline, Self-Refine, by **15.3 percentage points**.

Method	**Correct**	**Accuracy**
Zero-Shot CoT	192 / 1000	19.2%
ReAct	174 / 1000	17.4%
ToT	228 / 1000	22.8%
LATS	346 / 1000	34.6%
Self-Refine	391 / 1000	39.1%
ReTreVal	544/ 1000	**54.4%**

6.4 MMLU-Pro Results

To evaluate ReTreVal on broad multi-domain reasoning, we run all five baselines and ReTreVal on 1,000 MMLU-Pro problems using Gemini 2.5 Flash as the underlying LLM. This evaluation is label-free throughout: reference answers are never available to verification, backtracking, or memory updates and are used only for post-hoc accuracy measurement. MMLU-Pro is substantially harder than standard MMLU: each question has ten answer choices (versus four), and the problems span 14 subject areas ranging from quantitative fields (mathematics, physics, chemistry, engineering) to qualitative domains (law, history, philosophy, business). The ten-way choice reduces the random-baseline accuracy to 10%, making even the Zero-Shot CoT result of 19.2% a meaningful test of reasoning ability.

ReTreVal achieves **54.4%** accuracy (544/1000), substantially outperforming the best baseline (Self-Refine, 39.1%) by a margin of **+15.3 percentage points** (+39% relative). The gain is especially striking given that MMLU-Pro spans both quantitative and qualitative domains: the tool-augmented refinement loop provides verified computation for the quantitative subjects, while the tree exploration and backtracking benefit qualitative domains by considering alternative interpretations before committing to an answer.

6.5 Ablation Study

Table 5 quantifies the contribution of backtracking, self-rewriting memory, task decomposition, and tools on both benchmarks. MMLU-Pro ablations are label-free throughout. Backtracking is counted as active on a problem when validation rejects the initial synthesized answer and the system navigates to an unexplored sibling branch; problems whose initial answer passes validation directly are counted in the no-backtracking baseline.

Backtracking is the largest contributor on both benchmarks. It triggers on 22.0% of MATH-500 and 27.8% of MMLU-Pro problems, recovers 40.9% and 38.8% of them, and adds $+9.0$ and $+10.8$ pp, respectively; without it, MATH-500 accuracy (76.8%) falls below Self-Refine. On MMLU-Pro, rejection relies only on the self-critique and tool-consistency verifier, showing that label-free verification can identify when exploring an alternative branch is useful. **Disabling the self-rewriting memory** (no cross-problem learning) reduces accuracy by 8.6 pp on MATH-500 and 6.2 pp on

Table 5: Ablation of ReTreVal components on MATH-500 and MMLU-Pro (label-free). Backtracking is the largest contributor on both benchmarks. Decomposed vs. non-decomposed accuracies reflect selection of harder problems, not a causal effect. †Hard subset: Levels 4–5 on MATH-500; decomposed problems on MMLU-Pro. Cells marked — are reported for MATH-500 only.

Metric	MATH-500	MMLU-Pro
Backtracking		
Accuracy without backtracking	76.8% (384/500)	43.6% (436/1000)
Accuracy with backtracking	**85.8%** (429/500)	**54.4%** (544/1000)
Problems triggering backtracking	110/500 (22.0%)	278/1000 (27.8%)
Problems recovered by backtracking	45/110 (40.9%)	108/278 (38.8%)
Net accuracy gain	**+9.0 pp**	**+10.8 pp**
Memory		
Accuracy without memory	77.2% (386/500)	48.2% (482/1000)
Pass@3 without memory	82.4% (412/500)	—
Pass@5 without memory	83.8% (419/500)	—
Accuracy with memory (full ReTreVal)	**85.8%** (429/500)	**54.4%** (544/1000)
Net accuracy gain	**+8.6 pp**	**+6.2 pp**
Task decomposition		
Problems triggering decomposition	159/500 (31.8%)	197/1000 (19.7%)
Accuracy on decomposed problems	101/159 (63.5%)	145/197 (73.6%)
Accuracy on non-decomposed problems	295/341 (86.5%)	399/803 (49.6%)
Zero-Shot CoT accuracy on hard subset†	70.7%	68.0% (134/197)
ReTreVal accuracy on hard subset†	**76.4%**	**73.6%** (145/197)
Net gain on hard subset†	**+5.7 pp**	**+5.6 pp**
Tools		
Accuracy without tools	80.2% (401/500)	50.1% (501/1000)
Accuracy with tools (full ReTreVal)	**85.8%** (429/500)	**54.4%** (544/1000)
Net accuracy gain	**+5.6 pp**	**+4.3 pp**

MMLU-Pro, indicating that memory contributes beyond the structural search components, and **removing tools** costs 5.6 and 4.3 pp. **Task decomposition** is gated by the complexity estimator ($\geq 3/5$) and fires on 31.8% of MATH-500 and 19.7% of MMLU-Pro problems. Because it is invoked on the problems rated hardest, decomposed versus non-decomposed accuracy reflects selection rather than a causal effect. On MATH-500, decomposed problems concentrate on Levels 4–5 and score lower (63.5% vs. 86.5%), yet on Levels 4–5 ReTreVal reaches 76.4% versus Zero-Shot CoT's 70.7% ($+5.7$ pp). On MMLU-Pro, decomposed problems score higher (73.6% vs. 49.6%), and on those same problems ReTreVal exceeds Zero-Shot CoT (68.0%) by $+5.6$ pp, suggesting that sub-goal breakdown helps even on multiple-choice questions when the bottleneck is multi-step reasoning rather than knowledge recall.

6.6 Order Sensitivity

Problem order enters ReTreVal through cross-problem memory: although no future-problem information is available, the strategies retrieved for a problem depend on which earlier problems have been processed. We therefore rerun each full benchmark in its natural order and three independently shuffled orders, resetting memory before every run and keeping all other settings fixed. MATH-500 follows the ground-truth-assisted protocol of the main full-scale experiment; MMLU-Pro remains label-free. Because decoding is stochastic, the observed differences combine order sensitivity with run-to-run sampling variation and should not be read as a causal estimate of ordering alone.

On MATH-500, accuracy is 85.8% in natural order and 85.4%, 86.2%, and 84.8% under the three shuffles, a 1.4-point spread (seven problems); on MMLU-Pro it is 54.4% versus 53.9%, 54.9%, and 53.6%, a 1.3-point spread (13 problems). The natural-order score lies inside the shuffled range on both benchmarks, and reshuffling does not move accuracy consistently in either direction. These runs thus provide no evidence of a large sequence effect, although one stochastic run per ordering cannot establish order invariance or isolate memory-order effects from decoding variance.

6.7 Error Analysis

Aggregate accuracy hides qualitatively distinct failure and success modes. We manually inspected all 37 problems where ReTreVal is correct but Zero-Shot CoT is wrong (wins) and all 11 problems where ReTreVal is wrong but Zero-Shot CoT is correct (regressions); representative cases are detailed in Table 9 (Appendix G).

ReTreVal helps most when (i) an intermediate step is tool-verifiable and (ii) at least one sibling branch encodes a meaningfully different approach, and it hurts most when the complexity estimator over-budgets a trivially easy problem. Of the 11 regressions, 7 fall on Level 1–2 problems, strongly suggesting that tighter complexity calibration, or a short-circuit mechanism that accepts a highly confident single-shot answer without tree expansion, would recover most of the lost ground. The 3.4:1 ratio of wins to regressions confirms that the mechanism provides genuine error recovery rather than noise, while being transparent about the cases where additional machinery is a liability.

7 Discussion

Tool grounding is central to ReTreVal's advantage: offloading arithmetic and symbolic manipulation to verified solvers eliminates the computation errors that dominate baseline failures (strongest in Algebra and Number Theory). Typed-failure backtracking is qualitatively different from restarting: it navigates to a specific high-scoring sibling and injects structured failure context, steering the system away from the exact error class that failed. Gains concentrate on Level 4–5 problems; the system occasionally over-thinks Level 1–2 problems (7 of 11 regressions), indicating that tighter complexity calibration is the clearest avenue for improvement.

Practical use. ReTreVal is most useful for repeated problem-solving workflows, such as technical support, automated tutoring, scientific analysis, and engineering assistants. In these settings, typed failure traces make incorrect reasoning auditable, backtracking allows the system to recover before committing to an answer, and persistent memory retains recurring failure patterns across requests without modifying model weights. The label-free verifier enables these benefits when reference answers are unavailable, as in real deployments.

vs. frontier models. o1/o3 (OpenAI, 2024), DeepSeek-R1 (DeepSeek-AI, 2025), and rStar-Math (Guan et al., 2025) exceed 90% on MATH-500 via large-scale RL with process reward models (Lightman et al., 2024). ReTreVal is training-free and model-agnostic; the right comparison is *what does structured inference-time reasoning add to a given base model?*, a question the +7.2 pp gain over Self-Refine on the same 32B backbone directly answers.

Limitations. Computational overhead on hard problems may be prohibitive for latency-sensitive. Backtracking is restricted to root-level children; single-model critique may introduce correlated errors; keyword-Jaccard deduplication may miss semantically similar insights.

8 Conclusion

ReTreVal is the first inference-time framework to enable a fixed LLM to improve its problem-solving strategy across a benchmark sequence without any parameter update, through typed failure accumulation, self-rewriting memory, and skepticism-penalized scoring. It reaches **85.8%** with ground-truth-assisted verification and **84.2%** (421/500) with label-free verification on the full MATH-500 benchmark, and **54.4%** (544/1000) under label-free verification on MMLU-Pro. Across three full-scale reshuffles, the observed ranges are 84.8–86.2% on MATH-500 and 53.6–54.9% on MMLU-Pro; these narrow ranges show no large sequence effect in the tested runs, while not isolating ordering from stochastic decoding variance. Future directions include deeper backtracking with adaptive budget, embedding-based memory deduplication, decoupled critic models, and early exit for simple problems.

Reproducibility Statement

ReTreVal is training-free, so reproducing our results requires only the base models, the benchmarks, and the inference procedure. Section 4 specifies every component, including the scoring weights,

skepticism penalty, memory update, retirement, merge and retrieval rules, and the backtracking and decomposition triggers. Appendix A gives the complexity-dependent tree parameters (Table 6), the node fields (Table 7), pseudocode for tool-augmented refinement and backtracking (Algorithms 1 and 2), and the failure taxonomy; Appendices B and C describe the tool registry and its execution guards; and Appendix E lists the prompt templates. Section 5.1 reports the benchmarks, baseline configurations, sampling parameters, verification protocols, backbone models, and serving setup. Both benchmarks are public; reference answers are used only for scoring except where ground-truth-assisted verification is explicitly stated. Anonymized source code, including run and evaluation scripts, is provided as supplementary material.

AI USE STATEMENT

Generative AI (Claude, via Claude Code) was used in three ways. First, to assist with implementing and debugging the ReTreVal and baseline code; all code was reviewed and tested by the authors. Second, to help search related work and prepare citations; every cited source and every related-work claim was checked against the original publication by the authors. Third, to edit and revise this manuscript: condensing text to meet the page limit, restructuring and merging tables, drafting the Reproducibility Statement, and checking the arithmetic consistency of reported counts and percentages. All experiments were run by the authors, and every reported number comes from the authors' experimental runs; no results were generated by AI. All AI-assisted text was reviewed and revised by the authors, who set the research direction and final claims. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Jae-Woo Choi et al. ReAcTree: Hierarchical LLM agent trees with control flow for long-horizon task planning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2026. arXiv:2511.02424.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *International Conference on Machine Learning (ICML)*, 2023.

Google DeepMind. Gemini 2.5 flash. `https://deepmind.google/models/gemini/flash/`, 2025.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. rStar-Math: Small LLMs can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*, 2025.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173, 2023.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems*, volume 34, pp. 7294–7306, 2021.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pp. 22199–22213, 2022.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

LangChain AI. LangGraph: Build stateful multi-actor applications with LLMs. https://github.com/langchain-ai/langgraph, 2024.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

OpenAI. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Joon Sung Park, Joseph C O'Brien, Carrie J Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Timo Schick, Janice Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023a.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023b.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b.

Mert Yüksekgönül, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 55684–55708, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024. arXiv:2305.10250.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

A Methodology Details

Adaptive Tree Configuration

Table 6: Adaptive tree parameters by problem complexity.

Complexity	Max Depth	Children/Node
1–2 (Simple)	2	2
3 (Moderate)	3	3
4–5 (Complex)	4–5	4

Node Structure

Each node in the reasoning tree is a self-contained record that carries all information needed to score, refine, critique, and recover from it. Table 7 describes every field, its type, and its role in the framework.

Score evolution. A node's `combined_score` is recomputed after every refinement iteration as new tool observations arrive and after the cross-critique pass when siblings are also available. The frontier selector always picks the highest `combined_score` node that has not yet reached `d_max`, so score evolution directly drives which branch receives the next compute budget.

Algorithm: Tool-Augmented Refinement

Algorithm 1 Tool-Augmented Node Refinement

Input: node n, problem P, plan, memory summary M, tool registry $\mathcal{T}$, max steps S
Output: refined node n', execution trace τ
$\tau \leftarrow []$, calls $\leftarrow []$, errors $\leftarrow 0$
for step $s = 1$ to $S + 2$ **do**
  output $\leftarrow$ LLM(BuildPrompt(P, plan, n.thought, M, $\mathcal{T}$, τ))
  (thought, tool, input) $\leftarrow$ ParseOutput(output)
  if tool $=$ `FINISH` **then**
    n.thought $\leftarrow$ ExtractAnswer(input); **break**
  end if
  if IsLoop(tool, input, calls) **or** errors ≥ 3 **then**
    ForceFinish(n, τ); **break**
  end if
  obs $\leftarrow$ Dispatch($\mathcal{T}$, tool, input); τ.append({thought, tool, obs})
end for
return n, τ

Algorithm: Intelligent Backtracking

Failure Taxonomy

When a candidate answer fails critique or validation, the backtracking module classifies the failure before injecting it into the sibling branch. The taxonomy has two orthogonal axes: *failure*

Table 7: Fields stored per tree node and their roles in ReTreVal.

Field	Type	Role
`thought`	string	The current candidate solution text produced by the LLM. Updated after each refinement step; the final value is used as the node's answer.
`depth`	int	Distance from the root. Controls expansion budget: nodes at `depth` $\geq$ `d_max` are not expanded further, capping tree growth.
`refinement_count`	int	Number of refinement iterations applied to this node. Incremented each time the tool-augmented loop runs another LLM step; used by the absolute ceiling guard ($S + 2$).
`critique_history`	list[dict]	Ordered log of all critique results and injected failure contexts. During backtracking, the typed failure record (failure class, failed answer, error explanation) is appended here before the sibling branch resumes, structurally preventing the same error class from recurring.
`tool_trace`	list[dict]	Sequence of {thought, tool_name, input, observation} tuples from the refinement loop. Used by the loop-detection guard (exact-duplicate check) and retained for post-hoc analysis and error inspection.
`local_score`	float $\in [0, 1]$	Self-evaluation score: the LLM rates its own solution against the full reasoning trace. Weighted at 0.6 in the combined score because the self-evaluator has access to the complete tool trace.
`cross_score`	float $\in [0, 1]$	All-pairs pairwise critique score averaged over k siblings. Weighted at 0.4; provides an independent signal from branches that explored different strategies.
`combined_score`	float $\in [0, 1]$	$0.6 \times$ `local_score` $+ 0.4 \times$ `cross_score`, then multiplied by the skepticism penalty $(1 - \alpha(1 - r_a))$. This is the primary quantity used for frontier selection and backtracking target selection.
`children`	list[node]	Direct child nodes generated during tree expansion. The number of children is set by the complexity-adaptive branching factor (2 for simple, up to 4 for complex problems).
`parent_id`	string/None	Identifier of the parent node; `None` at the root. Used by the backtracking algorithm to locate root-level children (the set of high-level strategy branches) without traversing the full tree.

Algorithm 2 Intelligent Backtracking

Input: tree T, failed node n_f, explored set E, failure context F
Output: next node n_{next}, updated E, F'
$E \leftarrow E \cup \{\textsc{GetParentAfterRoot}(T, n_f)\}$
siblings $\leftarrow \textsc{GetRootChildren}(T) \setminus E$
if siblings $= \emptyset$ **then**
  return $\perp$
end if
$n_{\text{next}} \leftarrow \arg\max_{s \in \text{siblings}} s.\text{combined_score}$
$F' \leftarrow F \cup \{(\text{failure_type}, \text{failed_answer}, \text{error_analysis})\}$
Inject F' into n_{next}.critique_history
return n_{next}, E, F'

type (what went wrong) — `arithmetic_error`, `logic_error`, `incomplete_reasoning`, `wrong_constraint`, `off_by_one`, `missing_step` — and *approach type* (how the solution was structured) — `algebraic`, `arithmetic`, `pattern_matching`, `brute_force`. Both

labels are injected together so the sibling can simultaneously avoid the error class and the strategy family that produced it.

Injection format. The failure record injected into `critique_history` is a structured dict: `{failure_type, approach_type, failed_answer, error_explanation}`. The error explanation is a 1–2 sentence natural-language description generated by the failure-analysis prompt (Appendix E). The sibling's refinement prompt includes this record verbatim, allowing the LLM to condition its next attempt on both the structural label and the explanation.

B TOOL REGISTRY

ReTreVal's adaptive refinement loop dispatches to a benchmark-specific tool registry at inference time; the LLM selects whichever tool is appropriate for each step and irrelevant tools incur no cost. All tools are registered as plain Python callables with no internet or retrieval access, ensuring reproducibility and fair comparison with baselines.

MATH-500 tool registry (four tools targeting symbolic and numerical computation):

- **Symbolic Equation Solver** — algebraic manipulation, equation solving, and symbolic simplification via SymPy. Eliminates the algebraic errors that dominate baseline failures in Algebra and Number Theory.
- **Arithmetic Evaluator** — exact integer and floating-point computation via a sandboxed Python `eval` with a safe namespace. Verifies intermediate numerical results step-by-step.
- **Combinatorics Engine** — enumeration and counting via Python's `itertools` and `math` (permutations, combinations, factorials). Cross-checks closed-form answers against small brute-force cases on Counting & Probability problems.
- **LP / Optimisation Solver** — linear and mixed-integer programming via SciPy `linprog` / PuLP. Activated only when the LLM identifies an explicit constraint-satisfaction or optimisation structure in competition-level problems.

MMLU-Pro tool registry (six tools spanning quantitative and qualitative subjects): MMLU-Pro covers 14 subject areas, so the registry is broader. For quantitative subjects (mathematics, physics, chemistry, engineering) the Symbolic Equation Solver and Arithmetic Evaluator from the MATH-500 registry are reused unchanged. Four additional tools handle the domain diversity:

- **Unit Converter / Physical Constants** — SI unit conversion and physical constant lookup (speed of light, Planck's constant, etc.) via a static constants table. Used on physics and engineering problems where dimensional errors are a common failure mode.
- **Chemical Formula Evaluator** — molecular weight computation, stoichiometric balancing, and oxidation-state checks via a lightweight element table. Targets chemistry problems involving reaction arithmetic.
- **Statistical Calculator** — descriptive statistics, probability distributions (binomial, normal, Poisson), and hypothesis-test thresholds via SciPy `stats`. Used on economics, psychology, and data-science sub-questions.
- **Logic / Argument Checker** — propositional logic evaluation (truth-table generation, modus-ponens verification) for philosophy and formal-reasoning questions. Implemented as a small recursive Python evaluator over boolean ASTs.

For qualitative MMLU-Pro subjects (law, history, business, psychology) no computation tool is reliably applicable; in those cases the LLM emits `FINISH` without invoking any tool, and the gain comes entirely from tree exploration and backtracking rather than tool-augmented verification.

C TOOL-REFINEMENT ENGINEERING DETAILS

Loop Detection. To prevent infinite tool-calling cycles, the tool refinement executor implements three loop detection strategies. *Exact duplicate detection* terminates the loop with a forced `FINISH` if two consecutive tool calls share the same signature (tool name and first 80 characters of input). A

consecutive error threshold forces termination if three consecutive tool calls produce errors (indicated by "Error" prefixes in observations). Finally, an *absolute ceiling* of $S + 2$ LLM calls injects a forced-finish instruction, ensuring bounded computation regardless of execution behavior.

Output Parsing. The output parser extracts structured (thought, action, input) tuples from free-form LLM text using regex patterns. The parser handles the format `THOUGHT: ... ACTION: tool_name("arg")` with lenient fallbacks. If parsing fails entirely, the raw text is treated as a `FINISH` action, ensuring graceful degradation.

Per-Tool Error Handling. Each tool in the registry wraps its execution in a try/except block and returns a structured error string (prefixed `Error:`) rather than raising. This keeps the refinement loop alive: the LLM observes the error message as a regular observation and can choose a different tool or adjust its input on the next step. Tool-specific handling:

- **Symbolic Equation Solver (SymPy):** `SympifyError` and `TypeError` (malformed expression) return `Error: could not parse expression`. `NotImplementedError` (unsolvable system) returns `Error: solver could not find a closed-form solution` — the LLM typically falls back to numerical estimation.
- **Arithmetic Evaluator:** The `eval` sandbox restricts the namespace to `math`, `abs`, `round`, and basic operators; any attempt to access builtins or import modules raises `NameError`, returned as `Error: unsafe expression`. Division by zero returns `Error: division by zero`.
- **Combinatorics Engine:** Calls with non-integer or negative arguments to `math.comb`/`math.perm` return `Error: invalid arguments for combinatorial function`.
- **LP / Optimisation Solver:** Infeasible programs return `Error: problem is infeasible`; unbounded programs return `Error: problem is unbounded`. Both prompt the LLM to re-examine its constraint formulation.
- **Unit Converter / Physical Constants (MMLU-Pro):** Unknown unit strings return `Error: unrecognised unit`; the static constants table covers 120 NIST-standard values and returns `Error: constant not found` for anything outside it.
- **Chemical Formula Evaluator (MMLU-Pro):** Unrecognised element symbols return `Error: unknown element`. Unbalanceable reactions (stoichiometric inconsistency) return `Error: reaction cannot be balanced`.
- **Statistical Calculator (MMLU-Pro):** Out-of-domain distribution parameters (e.g., negative variance) return `Error: invalid distribution parameters`.
- **Logic / Argument Checker (MMLU-Pro):** Malformed propositional expressions return `Error: parse error in logical expression`. The evaluator only handles propositional logic; first-order quantifiers return `Error: quantified expressions not supported`.

Sandboxed Arithmetic Evaluator. The safe namespace passed to `eval` is constructed as:

```
safe_ns = {
    "__builtins__": {},
    "math": math,
    "abs": abs, "round": round, "min": min, "max": max,
    "pow": pow, "sum": sum, "int": int, "float": float,
}
```

This allows arithmetic and `math` library calls while blocking all I/O, imports, and attribute access on builtins. The expression is additionally stripped of the strings `import`, `open`, `exec`, and `eval` before parsing as a defence-in-depth measure.

D Full Computational Complexity Derivation

For a problem with complexity c, maximum depth $d_{\max}$, branching factor b, refinement step limit S, and backtracking attempts B, each iteration generates $O(b)$ children per expansion, and pruning to the top-k nodes keeps the maintained tree at $O(k \times d_{\max})$ nodes overall. Each node incurs up to S LLM calls plus tool executions for tool-augmented refinement, and the critique step adds $O(k^2)$ pairwise comparisons per expansion. Backtracking contributes an additional $O(B \times S)$ LLM calls for failure recovery, and decomposition adds $O(2m)$ sub-agent calls for m subtasks (up to four). The total LLM call complexity is therefore:

$$\mathcal{C}_{\text{LLM}} = O\Big(k \cdot d_{\max} \cdot (S + k + 2) + B \cdot S + 2m\Big)$$

The tool-augmented refinement loop is the dominant cost factor, but loop detection (Appendix C) and the absolute step ceiling ensure bounded execution per node. The backtracking overhead is controlled by the B parameter (default 1), and decomposition is triggered only for high-complexity problems, keeping average-case cost manageable.

E Prompt Templates

This appendix lists the two most novel prompt templates used by ReTreVal. Placeholders in `{curly braces}` are filled at inference time.

Failure Analysis (Backtracking). Invoked when a candidate answer fails validation. The analysis is injected into the sibling branch's context before refinement resumes.

```
The following solution is incorrect:

{failed_thought}

Ground-truth disagreement:
{validation_signal}

Classify the primary failure as one of:
  ARITHMETIC | LOGICAL | CONSTRAINT | MISREAD
Then explain in 1-2 sentences what specifically
went wrong and what to avoid in the next attempt.

FAILURE_CLASS:
EXPLANATION:
```

Task Decomposition Format. The structured subtask format emitted by the decomposer. Each subtask specifies a dependency list and a context field; `ATOMIC` is returned when the problem cannot be meaningfully split.

```
SUB_1: [description] | DEPENDS: none
       | CONTEXT: [info needed]
SUB_2: [description] | DEPENDS: sub_1
       | CONTEXT: [info needed]
```

Cross-Critique (Scoring) Prompt. Invoked once per expansion round to compare a node against each of its k siblings. The result contributes to `cross_score`.

```
Problem: {problem}

Solution A:
{solution_a}

Solution B:
{solution_b}
```

Table 8: Subject-wise accuracy (%) on MATH-500. Problem counts shown in parentheses. ReTreVal achieves the highest or tied-highest accuracy across all seven subjects.

Subject	ZS-CoT	ReAct	Self-Refine	ToT	LATS	ReTreVal
Algebra (124)	94.4	84.7	87.1	87.1	90.3	**97.6**
Number Theory (62)	80.6	83.9	87.1	82.3	83.9	**96.8**
Prealgebra (82)	91.5	76.8	85.2	86.6	86.6	**91.5**
Count. & Prob. (38)	71.1	71.1	81.6	71.1	81.6	**89.5**
Precalculus (56)	73.2	75.0	75.0	75.0	76.8	**85.5**
Intermed. Algebra (97)	56.7	58.8	63.9	63.9	57.7	**67.0**
Geometry (41)	51.2	48.8	63.4	63.4	56.1	**63.4**

```
Compare these two solutions to the problem above.
Which is more likely to be correct, and why?
Respond with:
  BETTER: A | B | TIE
  REASON: [1-2 sentences]
```

Memory Self-Rewrite Prompt. Invoked when the memory module detects that an approach type has failed more than `rewrite_threshold` times since it was last updated. The LLM rewrites the relevant strategy entry in-place.

```
You are updating your strategy memory after repeated failures.

Current strategy entry for approach type "{approach_type}":
{current_entry}

Recent failure contexts involving this approach:
{failure_contexts}

Rewrite the strategy entry to avoid these failure patterns.
Keep the entry under 3 sentences.

UPDATED_STRATEGY:
```

Complexity Estimation Prompt. Invoked once per problem before tree construction. The integer score (1–5) determines branching factor and max depth via Table 6.

```
Rate the reasoning complexity of the following problem
on a scale from 1 (trivial single-step) to 5 (requires
multi-stage reasoning, domain expertise, or constraint
satisfaction across multiple sub-goals).

Problem: {problem}

Respond with a single integer and a one-line justification.
COMPLEXITY: [1-5]
REASON: [one line]
```

F Subject-Wise Accuracy on MATH-500

Table 8 reports accuracy broken down by subject. ReTreVal achieves the strongest gains in algebraic subjects, reaching 97.6% in Algebra (+3.2 over Zero-Shot CoT) and 96.8% in Number Theory (+9.7 over Self-Refine), driven by tool-augmented computation eliminating mental-arithmetic errors. The largest absolute improvement is in Counting & Probability (+7.9 pp over Self-Refine/LATS at 81.6%). Intermediate Algebra (67.0%) and Geometry (63.4%) remain hardest for all methods, indicating opportunities for domain-specific tool extensions.

Table 9: Representative win and regression cases from the MATH-500 error analysis. Wins involve tool-verifiable intermediate steps; regressions concentrate on easy problems where over-thinking converts a correct single-shot answer into an incorrect explored alternative.

Outcome	Problem Type	What Happened
✓ Win	Number Theory, Level 4: modular arithmetic with a misapplied CRT step.	ZS-CoT commits to an incorrect Chinese Remainder setup and cascades the error. ReTreVal's critique flags a contradiction with a tool-verified residue check; backtracking selects a sibling branch that sets up the congruences correctly and recovers the right answer.
✓ Win	Counting & Probability, Level 3: count arrangements with a parity constraint.	ZS-CoT over-counts by a factor of 2. ReTreVal initially makes the same mistake, but the refinement loop queries a small-case enumeration tool whose output disagrees with the closed-form answer, triggering backtracking to a parity-aware sibling that matches ground truth.
✗ Regression	Prealgebra, Level 1: single-step fraction simplification.	ZS-CoT answers correctly in one shot. ReTreVal's complexity estimator overshoots, launches a depth-3 tree, and a spurious critique on a correct leaf drives backtracking into an incorrect sibling. The final answer regresses.

G Representative Error Cases

MATH-500 error cases. Table 9 shows representative cases from the manual inspection of all 37 wins (problems where ReTreVal is correct and Zero-Shot CoT is wrong) and all 11 regressions (problems where ReTreVal is wrong and Zero-Shot CoT is correct). Two patterns dominate the wins: tool-verified contradiction detection, where a tool observation disagrees with the LLM's intermediate value and triggers backtracking to a corrected sibling; and parity or constraint recovery, where the refinement loop queries a small brute-force case that exposes a miscounted closed-form answer. The regressions are concentrated at Level 1–2 and share a single root cause: the complexity estimator over-budgets a trivially easy problem, the tree expands unnecessarily, and a spurious cross-critique penalises the correct leaf in favour of a more elaborated but incorrect sibling.

MMLU-Pro error cases. Table 10 shows representative cases from MMLU-Pro, covering both quantitative subjects (where tools fire) and qualitative subjects (where gains come from tree exploration and backtracking alone).

Qualitative subject takeaway. The two MMLU-Pro regressions share a common root cause: on pure-recall questions with no reasoning chain to verify, the cross-critique mechanism can be misled by a confidently-stated wrong answer. This mirrors the Level 1–2 regression pattern on MATH-500 and suggests that a short-circuit rule — accepting the first answer on questions the complexity estimator scores ≤ 1 — would recover most of these cases without harming performance on harder questions.

Table 10: Representative win and regression cases from the MMLU-Pro analysis. Quantitative wins are driven by tool-verified computation; qualitative wins by backtracking to a different interpretive frame; regressions by over-expansion on recall-based questions.

Outcome	Subject / Type	What Happened
✓ Win	Chemistry: stoichiometry — balance a multi-step reaction and compute limiting reagent yield.	ZS-CoT makes an arithmetic error in the molar mass calculation, producing an off-by-one mole ratio. ReTreVal's Chemical Formula Evaluator returns the correct molecular weight; the critique detects the discrepancy and the refined node selects the correct answer choice.
✓ Win	Physics: dimensional analysis — identify the correct SI unit combination for a derived quantity.	ZS-CoT conflates two similar units. The Unit Converter tool verifies the dimensional equation and returns a mismatch signal; ReTreVal backtracks to a sibling that rechecks the base dimensions and selects the right choice.
✓ Win	Philosophy: formal argument validity — identify which of ten answer choices correctly identifies a logical fallacy.	No tool fires. ZS-CoT picks the most superficially plausible choice. ReTreVal expands two sibling branches with different interpretations of the argument structure; the cross-critique identifies that one branch correctly isolates the affirming-the-consequent pattern, and that branch wins the scoring round.
✓ Win	Law: statutory interpretation — determine which answer is consistent with a given legal rule and set of facts.	No tool fires. ZS-CoT confuses two answer choices that are superficially similar. ReTreVal's tree explores an alternative reading of the rule's scope; backtracking injects a `wrong_constraint` failure label that steers the sibling to re-read the statutory condition, recovering the correct answer.
✗ Regression	History: single-fact recall — identify the year of a specific historical event.	ZS-CoT answers correctly in one step. ReTreVal's complexity estimator scores the question at 2/5 and opens a shallow tree; a cross-critique spuriously penalises the correct answer because a sibling branch offers a plausible but wrong alternative year. The final aggregated score favours the wrong sibling.
✗ Regression	Business: definition recall — select the correct definition of a financial term.	ZS-CoT selects correctly. ReTreVal's second sibling branch, seeded with a `pattern_matching` approach, pattern-matches to a related but distinct term and receives a higher local score due to more detailed elaboration. The cross-critique cannot resolve the ambiguity and the wrong answer is selected.