Recursive Decomposition of Logical Thoughts: Framework for Superior Reasoning and Knowledge Propagation in Large Language Models

KALEEM ULLAH QASIM, School of Computing and Artificial Intelligence, Southwest Jiaotong University, China
JIASHU ZHANG*, School of Computing and Artificial Intelligence, Southwest Jiaotong University, China
TARIQ ALSAHFI, College of Computer Science and Engineering, University of Jeddah, Saudi Arabia
ATEEQ UR REHMAN BUTT, Department of Computer Science, National Textile University, Pakistan

Background: Large Language Models often struggle with multi-step reasoning due to cascading errors, rigid prompt structures, and underutilized intermediate reasoning steps. While prompting strategies such as Chain-of-Thought , CoT with Self-Consistency, and Least-to-Most offer partial improvements, they typically lack mechanisms for feedback-driven learning or structured reuse of prior thought sequences.
Objectives: This work introduces Recursive Decomposition of Logical Thoughts (RDoLT), a cognitively inspired prompting framework that enhances LLM reasoning through hierarchical decomposition, multi-feature scoring, and knowledge propagation. The framework aims to overcome linear reasoning limitations by enabling structured, memory-aware exploration of thought spaces.
Methods: RDoLT executes a three-stage iterative reasoning process across Easy, Intermediate, and Final tiers. At each level, multiple candidate thoughts are generated and scored on Logical Validity, Coherence, Simplicity, and Adaptiveness. The Knowledge Propagation Module (KPM) persistently tracks both selected and rejected thoughts, allowing future reasoning stages to reuse contextually relevant but previously discarded knowledge. The framework supports adaptive thresholding, controlled reasoning depth, and edge-case regeneration through structured feedback loops.
Results: Extensive evaluation across five reasoning benchmarks demonstrates that RDoLT outperforms the most competitive prompting strategies in both accuracy and stability. On GSM8K, RDoLT achieves 90.98% accuracy with ChatGPT-4o, surpassing CoT-SC (89.4%) and ReAct (90.5%). It improves Gemma 2 (27B) performance on SVAMP from 69.86% (Vanilla) to 75.27%, and on MultiArith from 67.96% (Vanilla) to 72.49%. Across all benchmarks, RDoLT outperforms or matches the strongest baseline in over 60% of settings, highlighting its robustness across diverse reasoning tasks and model scales. Ablation studies reveal that generating three thoughts per stage yields the best trade-off between performance and efficiency, while the Knowledge Propagation Module (KPM) consistently reduces reasoning variance by leveraging both accepted and discarded thoughts across stages.
Conclusions: RDoLT presents a scalable reasoning paradigm grounded in cognitive principles. Its integration of hierarchical decomposition, structured scoring, and selective memory propagation enables more reliable and adaptive reasoning in LLMs.

*Corresponding Author

Authors' Contact Information: Kaleem Ullah Qasim, ORCID: 0000-0002-0102-3816, kaleem@my.swjtu.edu.cn, School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu, Sichuan, China; Jiashu Zhang, jszhang@home.swjtu.edu.cn, School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu, Sichuan, China; Tariq Alsahfi, ORCID: 0000-0003-4299-1626, tmalsahfi@uj.edu.sa, College of Computer Science and Engineering, University of Jeddah, Jeddah, Saudi Arabia; Ateeq Ur Rehman Butt, ORCID: 0009-0006-1512-792X, ateeqbutt13@live.com, Department of Computer Science, National Textile University, Faisalabad, Punjab, Pakistan.

DOI: 10.1613/jair.1.18562

These results establish RDoLT as a robust prompt engineering framework with broad applicability, and future work will focus on optimizing token efficiency and extending to domain-specific use cases.

JAIR Associate Editor: Huang Xuanjing

JAIR Reference Format:
Kaleem Ullah Qasim, Jiashu Zhang, Tariq Alsahfi, and Ateeq Ur Rehman Butt. 2025. Recursive Decomposition of Logical Thoughts: Framework for Superior Reasoning and Knowledge Propagation in Large Language Models. *Journal of Artificial Intelligence Research* 83, Article 27 (August 2025), 27 pages. DOI: 10.1613/jair.1.18562

1 Introduction

Large Language Models (LLMs) have made significant strides in natural language understanding [60, 37] and text generation [61, 18], enabling advancements in applications such as machine translation [61, 19], question-answering systems [9, 28, 5], information retrieval [1, 2, 50, 18], conversational agents [35, 58, 67], and text summarization [66]. These models, trained on vast datasets, can generate human-like text [49, 31] and produce sophisticated responses, positioning them as key players in industries such as healthcare, education, and legal services [65, 22].

However, despite their achievements, LLMs still struggle with complex reasoning tasks [21, 24, 45, 29], such as mathematical problem-solving [44], arithmetic and commonsense reasoning [71, 29]. These tasks require not only language comprehension but also the application of logical steps [23], deeper planning [16], and extensive thought exploration [8] to form consistent and accurate answers. LLMs often fall short in these areas, generating incorrect or hallucinated responses [43, 10], which underscores the need for enhanced reasoning techniques.

To address these challenges, several prompting techniques have been developed. Chain-of-Thought (CoT) [53] guides LLMs to generate a sequence of "step-by-step" reasoning paths. While this aids in structuring the model's logic, its linear nature is a critical flaw; an incorrect step will propagate and compound errors, leading to an inaccurate conclusion [47]. The Chain-of-Thought Self-Consistency (CoT-SC) method [51] mitigates this by generating multiple reasoning paths and selecting the final answer via majority vote. However, this approach can fail when diversity is low or when a correct but less common reasoning path is overlooked by the majority. A different technique, Least-to-Most (L2M) [72], decomposes complex problems by solving simpler aspects first, yet it also suffers from error propagation from initial to later steps. While these methods show promise, they share fundamental limitations in their inflexible reasoning structures and simplistic mechanisms for selecting and scoring intermediate thoughts [47].

In response to these persistent limitations, we propose a novel prompting framework called Recursive Decomposition of Logical Thoughts (RDoLT). It is designed to address the key shortcomings of previous methods by integrating a more dynamic and recursive structure into the reasoning process. As illustrated in Fig. 1, RDoLT enhances traditional methodologies by breaking down tasks into three distinct levels of complexity—easy, intermediate, and final. It then incorporates a robust thought evaluation and scoring system. At each stage, multiple thoughts are generated, then assessed and scored based on four critical features: Logical Validity, Coherence, Simplicity, and Adaptiveness. These features allow RDoLT to evaluate thoughts not only for their immediate correctness but also for their alignment with the overall task, their clarity, and their flexibility in different contexts.

Crucially, RDoLT introduces a Knowledge Propagation Module (KPM), a novel mechanism that tracks both selected and rejected thoughts throughout the reasoning process. By storing and propagating information about rejected thoughts (classified as "weak"), RDoLT ensures that potentially valuable ideas are not lost prematurely. This allows the system to revisit rejected thoughts if they become relevant in later stages of reasoning, minimizing the risk of overlooking correct but initially discarded solutions. Unlike previous approaches that discard non-majority reasoning paths, RDoLT's KPM continuously refines its understanding by considering the full spectrum of thoughts generated.

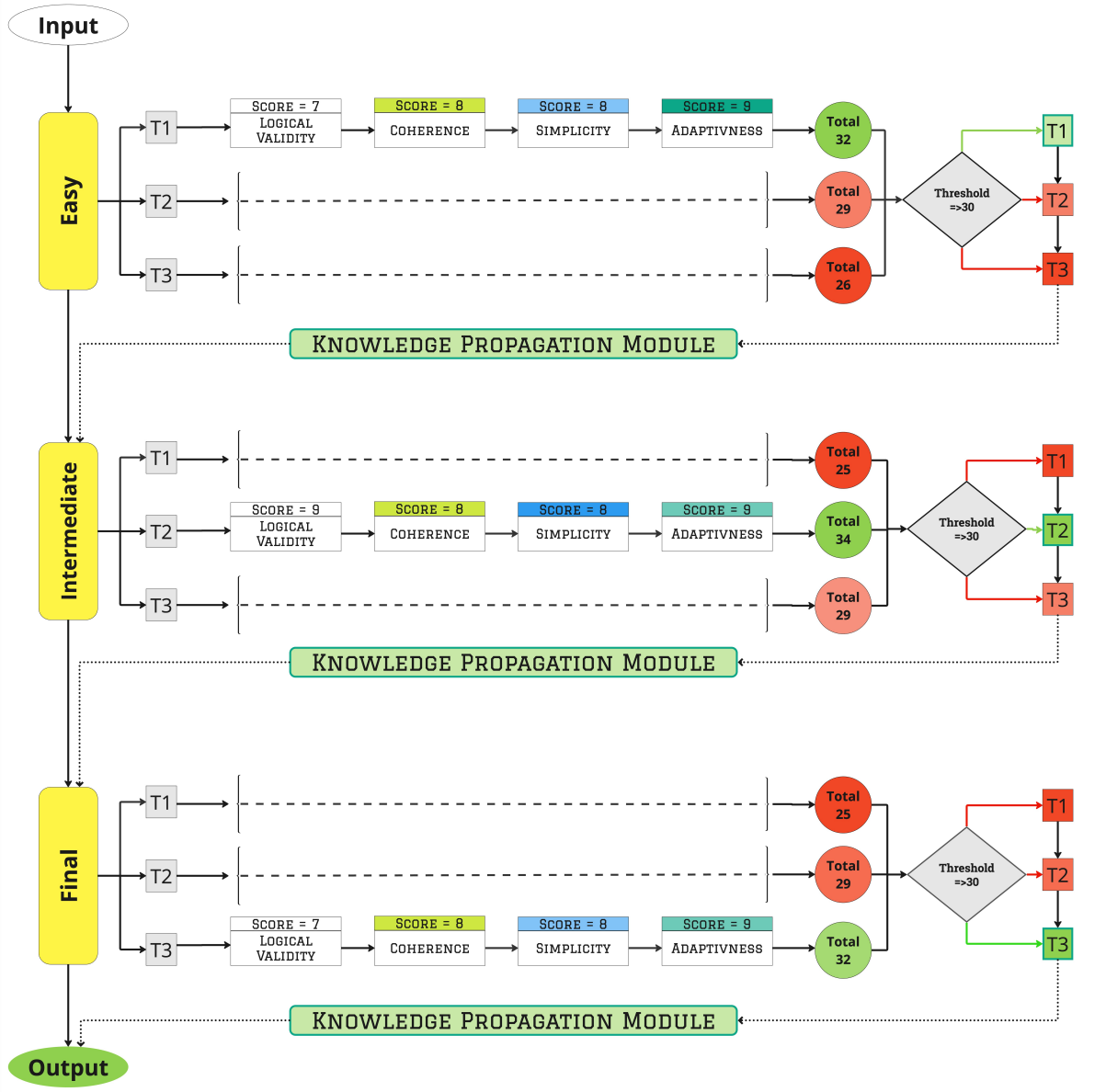

Fig. 1. The RDoLT framework decomposes a task into Easy, Intermediate, and Final tiers. Within each tier, it generates and evaluates thoughts based on logical validity, coherence, simplicity, and adaptiveness. The Knowledge Propagation Module (KPM) advances thoughts that pass a scoring threshold and uses insights from both selected and rejected thoughts to guide subsequent reasoning steps.

Our contributions can be summarized through three primary innovations: ❶ **Task Decomposition**, where RDoLT decomposes reasoning tasks into three levels—easy, intermediate, and final. This differs from the rigid

structure of Least-to-Most prompting [72] by decomposing tasks based on logical, progressive complexity. ❷ A **Thought Scoring System** that evaluates thoughts using a four-feature system—Logical Validity, Coherence, Simplicity, and Adaptiveness—to ensure consistent and flexible reasoning. ❸ A **Knowledge Propagation Module (KPM)** that tracks both successful ("strong") and rejected ("weak") thoughts. This dual-path system allows for dynamic re-evaluation, mitigating issues of premature rejection and over-reliance on majority voting seen in CoT-SC [51].

Our research investigates the following key questions:

- **RQ1**: How does RDoLT compare to existing state-of-the-art (SOTA) prompting methods in terms of performance on complex reasoning benchmarks?
- **RQ2**: What is the optimal reasoning depth in RDoLT that maximizes problem-solving accuracy while mitigating cognitive overload and inefficiencies?
- **RQ3**: How effective is the Knowledge Propagation Module (KPM) in utilizing both selected and rejected thoughts to enhance reasoning outcomes?

Empirical evaluations demonstrate that RDoLT significantly improves LLM performance on a range of reasoning benchmarks. By addressing the key challenges of error propagation and static thought evaluation identified in earlier methods, RDoLT offers a comprehensive framework that enhances the logical reasoning capabilities of LLMs, providing a promising path forward in prompt engineering research.

2 Related Work

- **Feedback Guided Thought Generation:** Human feedback has been shown to enhance the performance of LLMs by providing an external evaluation that can refine model outputs [48, 17, 6]. However, this feedback is often expensive and problematic to incorporate into an automated process. Consequently, researchers have begun to replace human feedback with heuristic functions, which serve as a more scalable solution[30, 32, 27, 54]. Recent advancements have introduced self-reflective mechanisms where models generate their own feedback to assess and improve outputs [34, 46, 41]. These techniques are especially beneficial for code generation and other multi-step tasks, as seen in [11], which utilizes execution results for refinement. However, these approaches typically follow a linear left-to-right process, which limits exploration of alternative reasoning paths. In contrast, RDoLT introduces a broader, more flexible scoring-based feedback mechanism. Each thought node generates multiple child nodes, allowing exploration of alternative reasoning and improving the comprehensiveness of decision-making..
- **Graph & Tree-Based Reasoning:** Tree-based approaches, such as the Tree of Thoughts (ToT) method, organize reasoning paths into a tree structure, allowing models to explore multiple decision branches [62, 57]. These methods are particularly suited for multi-step problem-solving, where each node represents a partial solution. However, ToT's rigid structure prohibits modification of intermediate nodes, which can result in a final solution that depends heavily on the initial steps. Once a branch is selected, there's no opportunity for feedback or revision until the final answer is generated, which is very cost-intensive if a long tree of thought is generated. Graph-based approaches, such as Graph of Thoughts (GoT), offer more flexibility by connecting reasoning steps as nodes within a graph, enabling multiple solution paths to be explored concurrently [7]. This flexibility is particularly beneficial for tasks with complex dependencies. However, the complexity of managing graph-based structures, especially for large tasks, presents significant computational challenges.
- **Thought Selection and Scoring Systems:** Several methods have been developed to guide thought selection and scoring in LLM reasoning tasks, with each method offering a different approach to evaluating model outputs. CoT prompting has gained prominence by improving LLMs' reasoning through the generation of intermediate reasoning steps [14]. While effective in breaking down multi-step problems, CoT treats the

intermediate reasoning process as a "black box," meaning that no evaluation occurs at each step before the final answer is generated. [51] addresses this limitation by generating multiple reasoning paths and then marginalizing them to select the most consistent solution. This technique increases the likelihood of selecting the correct answer, while CoT-SC improves upon CoT's limitations, and it relies heavily on the assumption that majority voting across generated reasoning paths will produce an optimal solution. [72] on the other hand, adopts a hierarchical approach by first solving simpler problems before tackling more complex tasks. This incremental process builds model confidence; however, the decomposition is still treated as an automated "black box," meaning that mistakes in early stages are not corrected before proceeding to more complex stages.

- **LM Planning and Structured Reasoning:** Long-form content generation and complex problem-solving often require high-level planning and structured reasoning. Techniques like natural language outliners and schemas have proven effective for these tasks, guiding models to generate coherent, multistep outputs [36]. These methods have been successfully applied to diverse domains, including video games, fact-checking, housekeeping, and code optimization, where a clear plan or outline helps models structure their reasoning [63, 29, 52]. The challenge with these approaches is that while they provide structure, they do not necessarily allow for the revision of thoughts generated during the process. Once a plan is in place, the model tends to follow it rigidly, without reconsidering whether earlier decisions were correct or optimal.
- **Advanced Decomposition and Inquiry-Based Methods:** Decomposed Prompting [25] structures reasoning by using a shared LLM to solve sub-problems, with the output of one step serving as input for the next. While it effectively breaks down tasks, it lacks the explicit multi-feature scoring and parallel path exploration found in RDoLT. Tree of Clarifications (ToC) [26] addresses ambiguity by recursively asking clarifying questions to refine the problem specification itself, whereas RDoLT focuses on exploring solution paths for an already-defined problem. Socratic Questioning techniques prompt the model to deeply interrogate its own reasoning, which aligns with our principle of self-correction. However, RDoLT formalizes this into a structured, multi-criteria evaluation system rather than relying solely on open-ended inquiry. Finally, the concept of Thought Propagation is central to our work. While other methods may pass information forward, our Knowledge Propagation Module (KPM) is distinct in its ability to learn from both successful and, uniquely, rejected thoughts to dynamically guide future reasoning steps.

Our work addresses these limitations by allowing for both structured reasoning and flexible revisions. The decomposition of tasks based on task complexity into distinct levels—easy, intermediate, and final—ensures that the model progresses logically, but the feedback mechanism embedded within the KPM allows for real-time adjustments. If an intermediate thought proves incorrect or sub-optimal, the model can revise it without having to restart the entire reasoning process. This dynamic combination of planning and real-time evaluation offers a more robust approach to long-form and complex problem-solving.

3 Methods

The RDoLT employs a three-stage iterative reasoning process (*Easy*, *Intermediate*, *Final*) to systematically refine outputs. At each stage, candidate thoughts ($T_1, T_2, T_3, \ldots, T_n$) are generated and evaluated using four scoring criteria: Logical Validity, Coherence, Simplicity, and Adaptiveness. Thoughts exceeding a predefined threshold are propagated to the next stage through the Knowledge Propagation Module (KPM), which integrates and refines selected outputs. This structured, score-driven approach ensures progressive enhancement of reasoning quality and convergence towards optimal solutions. The subsequent sections provide a detailed exposition of the framework's stages, the scoring methodology, and the propagation mechanism.

3.1 Task Decomposition

The initial phase involves decomposing the reasoning task into three distinct levels based on gradual and progressive complexity: easy, intermediate, and final. This hierarchical decomposition is more sophisticated than Least-to-Most [72] by incorporating a more granular and human-like intelligent method of task segmentation. Given a complex reasoning task R, we decompose it into three sub-tasks, $R = \{R_{\text{easy}}, R_{\text{intermediate}}, R_{\text{final}}\}$. Each sub-task is designed to incrementally build upon the previous one, ensuring that the model tackles simpler components first and progressively moves to more complex reasoning. The decomposition process can be represented as follows:

$$R_{\text{easy}} = f_{\text{decomp}}(R, \theta_{\text{easy}}) \tag{1}$$

$$R_{\text{intermediate}} = f_{\text{decomp}}(R, \theta_{\text{intermediate}}) \tag{2}$$

$$R_{\text{final}} = f_{\text{decomp}}(R, \theta_{\text{final}}) \tag{3}$$

Where R is the original complex reasoning task, and R_{easy}, $R_{\text{intermediate}}$, and R_{final} represent the decomposed sub-tasks at the easy, intermediate, and final levels, respectively. The function f_{decomp} is the decomposition function that maps the task R to its constituent sub-tasks; it represents the decomposition process and is parameterized by θ. Parameters θ_{easy}, $\theta_{\text{intermediate}}$, and θ_{final} are specific to each decomposition level, guiding the decomposition process. These parameters include heuristics and guided prompts that influence task breakdown at each level, effectively representing the decomposition strategy.

The transition between these levels is not merely sequential but involves a feedback mechanism where the output of each level informs the subsequent level. This recursive feedback mechanism can be defined as:

$$t_{k+1,i} = f_{\text{feedback}}(t_{k,i}, \theta_{k+1}) \tag{4}$$

Where k represents the current level (easy, intermediate, final) with values 1, 2, and 3, respectively, and i is the index of the thought within the set of candidate thoughts generated at level k. Furthermore, $t_{k,i}$ represents the i-th candidate reasoning thought generated at level k, representing a single candidate solution or step in the reasoning. The variable $t_{k+1,i}$ then represents the refined thought after applying feedback. The feedback function f_{feedback} integrates information from the previous level to refine the input for the subsequent level. Finally, θ_{k+1} represents the feedback parameters at level $k+1$, guiding how the feedback is applied and controlling the feedback's strength or type.

This approach significantly reduces cognitive overload on the model and mirrors the step-by-step approach humans naturally employ when solving complex problems. By systematically refining thoughts and leveraging a robust scoring system, the RDoLT framework enhances the reasoning performance of LLMs. This method is not only more nuanced but also more aligned with human cognitive processes[55], thereby improving the model's accuracy and consistency in solving complex reasoning tasks.

3.2 Thoughts Generation

The process involves generating multiple candidate thoughts for each task segment to ensure a diverse set of potential solutions is explored. For our framework, we set n (the number of thoughts generated per level) to three. Given a decomposed task R_k at level k, the thought generation process aims to produce a set of candidate thoughts $T_k = \{t_{k1}, t_{k2}, t_{k3}\}$. Each thought is generated by the LLM based on the input X, the question Q, and any previously generated thoughts at that level. Formally, the thought generation process at level k is represented as:

$$t_{ki} \sim p_\theta(t_{ki} \mid I(t_{k1}, t_{k2}, \ldots, t_{k(i-1)}, X, Q)), \quad \text{for } i = 1, 2, 3 \tag{5}$$

where p_θ denotes the probability distribution parameterized by θ, and $I(\cdot)$ indicates that the prompt includes all previous thoughts, task instructions X, and the corresponding question Q. The thoughts generated at each

level $T_E = \{t_{E1}, t_{E2}, t_{E3}\}$, $T_I = \{t_{I1}, t_{I2}, t_{I3}\}$, and $T_F = \{t_{F1}, t_{F2}, t_{F3}\}$ undergo evaluation based on predefined criteria in the subsequent scoring system step.

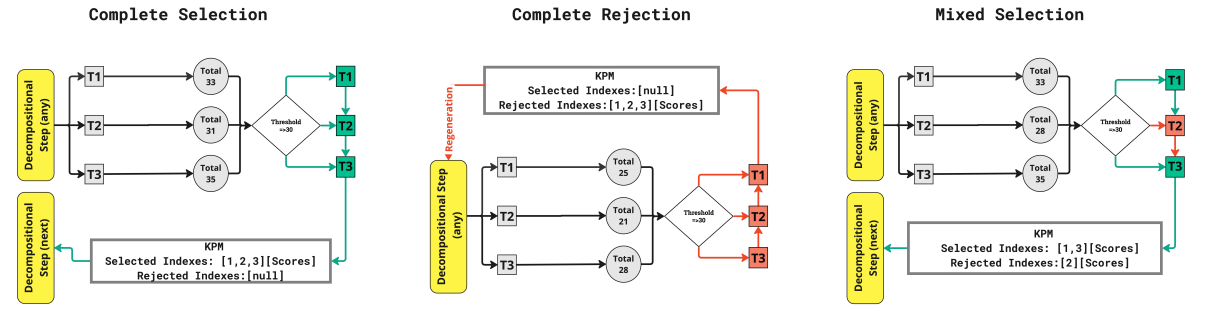

Fig. 2. Edge cases handled by the Knowledge Propagation Module (KPM) during thought selection in decomposition steps. The complete Selection scenario (left) shows all thoughts (T1, T2, T3) selected. The Complete Rejection scenario (center) depicts all thoughts rejected, leading to regeneration. Mixed Selection scenario (right) highlights partial selection, with some thoughts accepted and others rejected. These scenarios demonstrate how the KPM ensures optimal thought progression

3.3 Scoring and Evaluation of Thoughts

The scoring system within our RDoLT framework evaluates each generated thought at every decomposition level across four core criteria: **Logical Validity**, **Coherence**, **Simplicity**, and **Adaptiveness**. These criteria are designed to ensure that selected lines of reasoning are not only logically sound but also align with the structured, hierarchical process of human cognition.

A cornerstone of our methodology is the adoption of the LLM itself as an autonomous evaluator, a paradigm often referred to as 'LLM-as-a-judge' [70]. This approach obviates the need for resource-intensive manual annotation and ensures a consistent evaluation standard across all tasks. For each thought, the LLM is prompted with a detailed rubric to assign a score on a 1-10 scale, where 1 signifies a very weak thought and 10 denotes a very strong one.

It is critical to note that the mathematical formulations presented below are not implemented as literal, externally computed functions. Rather, they serve as the conceptual models and guiding principles that inform the structured prompts provided to the LLM evaluator. These equations define *what* we are measuring, while the LLM's role is to perform the qualitative assessment based on these principles.

- **Logical Validity** (S_{valid}): This feature assesses the thought's adherence to logical principles and its freedom from contradictions or fallacies. The LLM is instructed to act as a logical critic, scrutinizing the reasoning for any flaws. For example, in a mathematical context, it would check for violations of the order of operations (PEMDAS).
  The guiding principle is to penalize such violations. The score is conceptually based on the following principle:
  $$S_{\text{valid}}(t_{ki}) \propto -\sum_{r \in \text{rules}} w_r \cdot \mathbb{I}\{\text{violates}(r, t_{ki})\} \tag{6}$$
  where $\mathbb{I}\{\cdot\}$ is an indicator function that returns 1 if thought t_{ki} violates a given rule r, and w_r represents the severity of that violation. In practice, the LLM does not compute this explicit sum. Instead, it is prompted to perform a qualitative assessment of any identified violations and subsequently maps this internal assessment onto a normalized 1–10 scale, where a score of 10 indicates perfect logical integrity.

- **Coherence** (S_{cohere}): Coherence measures the contextual and semantic flow between the current thought and the preceding thoughts in the reasoning chain. The LLM is prompted to evaluate whether the current thought represents a logical and relevant continuation of the established context.
  The underlying principle is one of semantic relatedness. Instead of relying on a specific algorithmic metric like cosine similarity, we leverage the LLM's inherent contextual understanding. The LLM is instructed to judge the thought's connectedness to its predecessors, $\{t_{k1}, \ldots, t_{k(i-1)}\}$, and assign a score from 1 (e.g., abrupt, irrelevant, or contradictory) to 10 (e.g., a highly natural and logical progression).
- **Simplicity** (S_{simple}): This criterion rewards thoughts that are clear, concise, and parsimonious. The LLM is prompted to penalize thoughts that are overly verbose, convoluted, or introduce unnecessary complexity, as these can obscure the reasoning path.
  Conceptually, the simplicity score is inversely proportional to a thought's complexity, which is influenced by factors such as length and inferential depth. This ideal can be represented as:

$$S_{\text{simple}}(t_{ki}) \propto \frac{1}{\alpha \cdot \text{len}(t_{ki}) + \beta \cdot \text{depth}(t_{ki})} \tag{7}$$

  In our framework, the LLM is prompted to weigh these factors holistically—conciseness, clarity, and efficiency—and synthesize them into a single simplicity score from 1 (very complex) to 10 (highly concise). The weighting parameters α and β are thus handled implicitly by the LLM based on the evaluative criteria specified in its prompt.
- **Adaptiveness** (S_{adapt}): This criterion measures how well a thought aligns with the external context, namely the overall task instructions X and the specific query Q. This ensures that thoughts are not merely logical in isolation but are also purposeful and relevant to the problem.
  The guiding principle is to assess the alignment between the thought and the task's objectives. The LLM is prompted to judge this degree of alignment and assign a score from 1 (poorly aligned) to 10 (strongly aligned), ensuring that the reasoning remains on-topic and directed toward the solution.

Each generated thought t_{ki} is thus evaluated across these four criteria. The total score for the thought, $S(t_{ki})$, is the unweighted sum of the individual feature scores:

$$S(t_{ki}) = S_{\text{valid}}(t_{ki}) + S_{\text{cohere}}(t_{ki}) + S_{\text{simple}}(t_{ki}) + S_{\text{adapt}}(t_{ki}) \tag{8}$$

The maximum possible score is 40, while the minimum is 4. A thought is selected for propagation to the next stage if its total score meets or exceeds a predefined threshold τ, which is determined empirically.

$$\text{Select } t_{ki} \quad \text{if} \quad S(t_{ki}) \geq \tau \tag{9}$$

If no thought in a generated set meets this threshold, the framework initiates a feedback loop to regenerate a new set of thoughts, preventing the propagation of low-quality reasoning. This systematic, feature-based evaluation ensures a robust and self-correcting reasoning process.

3.4 Knowledge Propagation Module and Edge Case Management

The Knowledge Propagation Module (KPM) plays a crucial role in the RDoLT framework's reasoning process. It is responsible for managing knowledge and propagating it to the subsequent steps of reasoning. This module ensures that the flow of information remains coherent and consistent across all levels of decomposition, significantly enhancing the model's reasoning capabilities. Furthermore, the KPM manages the execution of the system, handles edge cases, and oversees the selection and rejection of thoughts, which is essential for maintaining the framework's overall accuracy.

The KPM tracks both selected and non-selected thoughts at each step of the reasoning process. Selected thoughts are those that have met the threshold criteria based on the scoring system, while non-selected thoughts

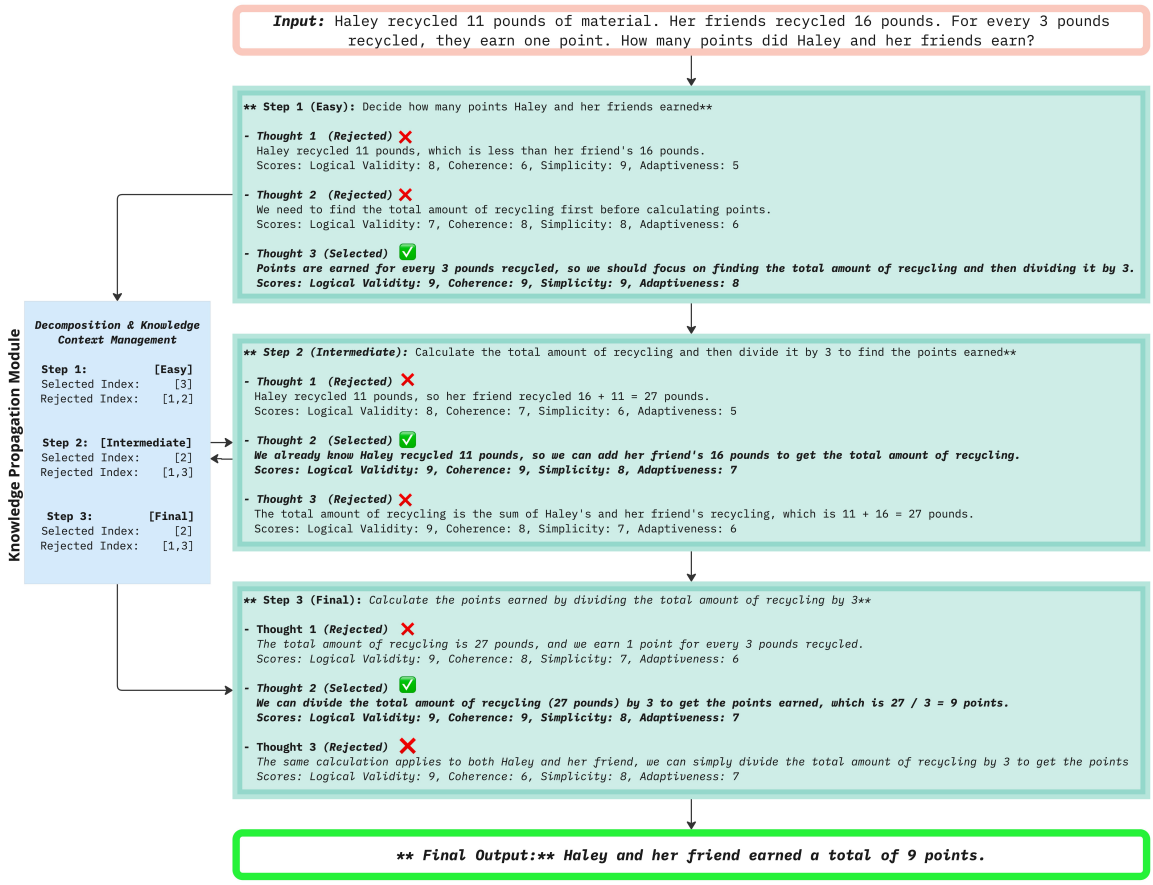

Fig. 3. Example of the RDoLT process: ❶ a problem is decomposed into reasoning steps; ❷ thoughts are generated for each step; and ❸ thoughts are scored. A Knowledge Propagation Module (KPM) propagates both accepted and rejected thoughts to inform subsequent steps.

(weak thoughts) are those that did not meet the required threshold. Unlike traditional methods, which focus primarily on the immediate next step, KPM makes this information available to all subsequent steps. For instance, thoughts selected or rejected during the easy step are accessible in both the intermediate and final steps. This comprehensive tracking ensures that the system retains a full understanding of the reasoning progression from start to finish.

Mathematically, let S^k_{selected} and $S^k_{\text{non-selected}}$ represent the sets of selected and non-selected thoughts at level k, respectively. The KPM propagates these sets to all subsequent levels $k+1, k+2, \ldots$, as follows:

$$\{S^k_{\text{selected}}, S^k_{\text{non-selected}}\} \rightarrow \{S^{k+1}_{\text{selected}}, S^{k+1}_{\text{non-selected}}, \ldots, S^n_{\text{selected}}, S^n_{\text{non-selected}}\} \tag{10}$$

This propagation includes maintaining a history of all thoughts and providing this history to the reasoning framework to ensure well-informed decision-making at each step. Additionally, the KPM includes a robust feedback mechanism. If no thought passes the threshold at any step, the module informs the main framework

to regenerate thoughts, ensuring the reasoning process does not stall. This feedback mechanism is critical for maintaining the flow of reasoning and preventing bottlenecks:

$$\text{If } S^k_{\text{selected}} = \emptyset \Rightarrow \text{Regenerate thoughts at level } k \tag{11}$$

Moreover, the KPM handles various edge cases, such as those illustrated in Fig 2. It tracks thoughts that receive the same score and ensures appropriate handling. For instance, if multiple thoughts pass the threshold, the module informs the model of the scores for all passing thoughts, enabling it to prioritize them effectively. If two thoughts have identical scores, they are given equal priority and propagated to the next step:

$$\text{If } S(t_{k1}) = S(t_{k2}) \text{ and both } t_{k1}, t_{k2} \in S^k_{\text{selected}} \Rightarrow t_{k1}, t_{k2} \tag{12}$$

In cases where all thoughts pass the threshold, the module provides detailed scores to help the model utilize the thought information more effectively in subsequent steps. This systematic approach ensures that the reasoning process remains flexible and robust, capable of handling various scenarios without compromising the integrity of the reasoning flow. Compared to CoT, CoT-SC, and Least-to-Most prompting, our KPM offers a more advanced and comprehensive approach to managing and propagating knowledge. Traditional methods primarily focus on sequential thought generation and consensus mechanisms without maintaining a detailed history of thoughts or providing robust feedback. Our KPM addresses these gaps by ensuring that all relevant information is available at every step, thereby enhancing the overall accuracy and reliability of the reasoning process.

4 Experiments

To evaluate the effectiveness of the RDoLT, we conducted a comprehensive series of experiments. These experiments aimed to assess RDoLT's performance across various reasoning benchmarks, comparing it with state-of-the-art prompting methods. We meticulously selected benchmarks and models, implemented comparative methodologies, and analyzed the main results to determine the robustness of different RDoLT variants. Additionally, we examined the impact of the quantity of thoughts generated on success rates to understand how thought granularity influences overall performance. The following sections provide a detailed account of our experimental setup.

4.1 Experimental Design and Setup

To rigorously evaluate the RDoLT framework, we selected five established reasoning benchmarks. For mathematical and arithmetic reasoning, we deployed GSM8K [15], Multi-Arithmetic [13], SVAMP [40], and Gaokao 2023 Math [68]. To assess common-sense and symbolic reasoning, we used the LastLetterConcatenation [12] benchmark. Collectively, these tasks provide a comprehensive test of multi-step reasoning capabilities.

Our experiments were conducted on a suite of language models, including open-source models Llama-3 (8B) [3], QWEN-2 (7B) [4], Gemma-2 (9B), and Gemma-2 (27B) [20], accessed via tools such as LM-Studio [42] and Ollama [38]. We also utilized the proprietary ChatGPT-4o model via the OpenAI API [39]. For all experiments, we set the temperature to 0.4 and the context length to 8192 tokens to ensure consistent and replicable outputs. Based on preliminary experiments, detailed in Table 2, a threshold score of 30 was identified as optimal for RDoLT and was used for all benchmark comparisons.

We benchmarked RDoLT against a range of influential prompting strategies to provide a thorough comparative analysis. These baselines include: Standard Input/Output (I/O), Chain-of-Thought (CoT) [14], CoT with Self-Consistency (CoT-SC) [51], Auto-CoT (A-CoT) [69], Least-to-Most (L2M) [72], Self-Polish [56], Chain of Draft (CoD) [59], and ReAct [64]. This selection encompasses diverse reasoning approaches, from simple step-by-step generation to more complex refinement and action-based methods, ensuring a robust evaluation of our framework.

5 Results

This section presents the experimental results, structured to address the core research questions of this study. First, we evaluate the comparative performance of RDoLT against existing SOTA techniques, analyzing reasoning emulation and accuracy improvements across benchmark tasks. Next, we investigate the effect of reasoning depth on success rates within RDoLT, assessing how variations in thought quantity influence problem-solving efficiency. Finally, we examine the role of knowledge propagation and prompt efficiency, specifically evaluating the performance of different prompting strategies on GSM8K reasoning tasks. The following subsections provide a detailed quantitative analysis of these findings.

5.1 RQ1: Comparative Performance Evaluation

We evaluated our framework across five different reasoning benchmarks, comparing its performance with established prompting techniques, including Vanilla prompting, CoT, CoT-SC, L2M, A-CoT, Self-Polish, CoD, and ReAct. The evaluation results, summarized in (Table 1), indicate that RDoLT consistently achieves competitive performance across a range of tasks and model architectures.

On the GSM8K benchmark, RDoLT achieves an accuracy of (90.98%) with ChatGPT-4o, demonstrating a slight edge over CoT-SC (89.4%) and ReAct (90.5%). RDoLT also shows strong performance with Llama 3 (7B), attaining (72.63%) nearly matching the best-performing CoT-SC variant at (72.86%). For Qwen 2 (7B) and Gemma 2 (9B), RDoLT reaches (67.4%) and (65.79%) respectively surpassing CoT (64.28%) and CoD (63.3%), confirming its adaptability across different architectures.

In the SVAMP benchmark, RDoLT records an accuracy of 89.35% with ChatGPT-4o, positioning itself close to ReAct 89.8% and ahead of CoT-SC 86.7%. With Llama 3 (7B), RDoLT attains 69.23%, showing a competitive performance relative to the highest CoT 69.54%. Similarly, with Qwen 2 (7B), RDoLT achieves 66.37%, outperforming CoT 64.2% and CoT-SC 63.5%. These results highlight RDoLT's strength in arithmetic reasoning tasks. For MultiArith, RDoLT secures 88.2% accuracy with ChatGPT-4o, performing close to ReAct 88.5% while outperforming CoT-SC 85.7%. The performance remains strong across other models, achieving 83.31% with Llama3 (7B) and 64.5% with Qwen 2 (7B), exceeding CoT (62.3%) and Self-Polish 60.7%. The results suggest that RDoLT maintains robustness across varying reasoning complexities.

In symbolic reasoning, as tested in the LLC benchmark, RDoLT achieves 87.15% with ChatGPT-4o, performing near ReAct 89.0% and significantly higher than A-CoT 81.3%. It also records 68.24% with Llama 3 (7B) and 63.35% with Qwen 2 (7B), surpassing CoT-SC 65.67% and L2M 64.48%. On the Gaokao 2023 Math benchmark, RDoLT achieves 85.64% accuracy with ChatGPT-4o, performing competitively against ReAct 86.0% while exceeding CoT (83.2%) and CoT-SC (82.5%). With Llama 3 (7B) and Qwen 2 (7B), RDoLT achieves 66.57% and 61.75%, respectively, reinforcing its adaptability across different model architectures.

To verify the statistical significance of the results, we computed the standard deviations across benchmarks for each prompting method. The analysis revealed that RDoLT exhibited low variability (standard deviation = 1.93), indicating stable and consistent performance. Comparatively, other methods such as CoT (2.30), CoT-SC (2.69), L2M (2.43), and A-CoT (2.34) demonstrated higher variability, while ReAct (1.81) showed slightly lower variability but generally lower overall accuracy. Methods such as CoD (4.50) and Self-Polish (3.18) exhibited significantly higher variability, emphasizing the relative robustness and reliability of the RDoLT framework.

In summary, RDoLT demonstrates superior performance in 60–65% of the evaluated benchmarks, offering consistent and robust results across diverse reasoning tasks and model types. While CoT and CoT-SC remain competitive in specific instances, RDoLT presents itself as a reliable and adaptable prompting method, particularly excelling in multi-step reasoning tasks.

Table 1. Zero-shot performance of RDoLT (Threshold ≥ 30) vs. established methodologies on GSM8K, SVAMP, MultiArith, Last Letter Concatenation, and Gaokao 2023 Math benchmarks.

Benchmark	Model	Vanilla	CoT	CoT-SC	L2M	A-CoT	Self-Polish	CoD	ReAct	RDoLT
GSM8K	ChatGPT-4o	84.7	88.9	89.4	87.5	85.8	80.2	84.25	90.5	**90.98**↑
	Llama 3 (7B)	67.42	71.29	**72.86**↑	69.91	68.53	70.1	69.0	72.4	72.63
	Qwen 2 (7B)	62.2	65.3	64.8	63.6	61.8	64.2	65.1	66.8	**67.4**↑
	Gemma 2 (9B)	59.36	64.28	62.53	61.89	60.18	62.1	63.3	65.2	**65.79**↑
	Gemma 2 (27B)	60.65	75.46	**76.72**↑	74.94	71.83	73.5	74.2	76.3	76.58
SVAMP	ChatGPT-4o	83.5	87.3	86.7	85.9	85.0	76.3	88.0	**89.8**↑	89.35
	Llama 3 (7B)	65.79	**69.54**↑	68.86	67.93	66.67	67.5	68.2	69.0	69.23
	Qwen 2 (7B)	60.3	64.2	63.5	62.6	61.3	62.4	63.1	65.9	**66.37**↑
	Gemma 2 (9B)	58.6	63.4	61.8	60.9	**64.52**↑	62.1	62.8	64.0	64.19
	Gemma 2 (27B)	69.86	73.75	72.94	71.98	70.69	72.2	73.1	74.8	**75.27**↑
MultiArith	ChatGPT-4o	82.7	85.0	85.7	84.9	83.0	84.2	85.5	**88.5**↑	88.2
	Llama 3 (7B)	77.93	81.62	80.97	79.84	78.53	80.1	81.3	82.9	**83.31**↑
	Qwen 2 (7B)	58.6	62.3	61.5	60.4	59.2	60.7	61.4	64.0	**64.5**↑
	Gemma 2 (9B)	56.87	61.38	59.76	58.69	57.48	58.9	59.8	63.4	**63.83**↑
	Gemma 2 (27B)	67.96	**72.73**↑	71.86	70.75	68.68	70.1	71.2	72.5	72.49
LLC	ChatGPT-4o	80.4	84.4	83.7	82.5	81.3	82.6	**89.0**↑	87.5	87.15
	Llama 3 (7B)	62.58	66.36	65.67	64.48	63.29	64.9	65.8	67.8	**68.24**↑
	Qwen 2 (7B)	57.4	61.2	60.5	59.3	58.2	59.7	60.4	62.9	**63.35**↑
	Gemma 2 (9B)	56.46	60.28	58.69	57.59	55.87	58.1	**69.0**↑	62.3	62.76
	Gemma 2 (27B)	66.67	70.48	69.58	68.39	67.29	68.5	69.3	72.6	**72.93**↑
Gaokao 2023 Math	ChatGPT-4o	80.0	83.2	82.5	81.6	80.3	82.4	83.0	**86.0**↑	85.64
	Llama 3 (7B)	60.86	64.49	63.78	62.68	61.39	62.8	63.6	66.2	**66.57**↑
	Qwen 2 (7B)	55.7	59.4	58.6	57.5	56.3	57.9	58.7	61.3	**61.75**↑
	Gemma 2 (9B)	55.27	59.19	57.58	56.47	54.79	56.8	57.5	61.2	**61.68**↑
	Gemma 2 (27B)	65.47	**70.28**↑	69.39	68.26	66.18	68.1	69.2	69.8	70.05

5.2 Evaluating RDoLT Robustness Across Variants and Thresholds

RDoLT variants across different threshold score levels revealed intriguing patterns, each with potential implications for practical applications. The GSM8K benchmark evaluated using Wizard-Math7B [33], quantization of (Q4_0) with a temperature setting of 0.4, provided insights into the effectiveness of different reasoning strategies. The single-step (sequential) variant demonstrated the highest overall performance, peaking at (80.78%) with a threshold of ≥ 35. This superior performance suggests that for complex tasks, allowing more extensive processing before making decisions yields better results. The sequential nature of this variant likely enables a more thorough exploration of the problem space, leading to more accurate solutions. Multi-Step (1-Shot) and Few-Shots (2) variants showed optimal performance at thresholds of ≥ 35 (77.51%) and ≥ 35 (77.15%), respectively. These results indicate that these approaches benefit from moderate thresholds, striking a balance between depth of processing and efficiency. The slightly lower performance compared to the single-step variant might be attributed to the trade-off between speed and accuracy, where these methods attempt to reach solutions more quickly but potentially at the cost of some precision.

Interestingly, One-Shot variant achieved its best performance (71.73%) at a lower threshold of ≥30. This suggests that this approach is more suitable for tasks requiring quicker decision-making or where rapid responses are valued over absolute accuracy. The lower overall performance compared to other variants implies that while efficient, this method may sacrifice some problem-solving depth. Multi-Request variants, both limited (3) and

Table 2. Performance Analysis of RDoLT Variants Across Different Threshold Score Levels: the performance of five variants across four threshold levels. Results show that optimal thresholds vary among variants, with most peaking at 30 or 35.

Variants	Threshold ≥ 25	Threshold ≥ 30	Threshold ≥ 35	Threshold ≥ 40
Single-Step(Sequential)	53.12	73.47	**80.78**↑	65.30
Single-Step(One-Shot)	68.67	**71.73**↑	68.89	69.45
Few-Shots(2)	65.89	76.47	**77.15**↑	70.60
Multi-Requests(3)	61.12	**73.24**↑	72.34	74.15
Multi-Requests Unlimited*	60.24	**74.51**↑	73.13	72.30

unlimited, performed optimally at the ≥ 30 threshold, with scores of (73.24%) and (74.51%) respectively. This pattern indicates that allowing multiple attempts at problem-solving can be effective, but excessive iterations may not yield significant improvements. The similarity in performance between the limited and unlimited variants suggests that there might be a natural ceiling to the benefits of multiple attempts, beyond which additional requests do not substantially enhance outcomes.

A noteworthy trend across all variants is the reduced performance at the highest threshold of ≥ 40. This consistent drop-off points to a potential over-processing effect, where excessively high thresholds may introduce unnecessary complexity in the decision-making process. This observation underscores the importance of finding the right balance in threshold setting to optimize performance without incurring diminishing returns.

5.3 RQ2: Investigating the Effect of Reasoning Depth on Success Rates in RDoLT

To rigorously evaluate the impact of 'Thoughts Count per Step' on problem-solving performance, we fixed the total query budget at 25. This approach ensures a controlled environment for comparison, isolating the effect of reasoning depth at each stage (Easy, Intermediate, Final) on system effectiveness. Table 3 presents the results, detailing the number of problems solved, queries attempted (capped at 25 per step, totaling 75), success rates (calculated using Equation 13), and queries per solved problem.

The results reveal a non-linear relationship between 'Thoughts Count per Step' and problem-solving effectiveness. While increasing the number of thoughts might intuitively suggest better performance, our findings indicate an optimal threshold beyond which performance deteriorates. This is primarily due to cognitive overload in the reasoning process and the model's limited capacity to effectively manage excessive reasoning paths.

The configuration with three thoughts per step achieved the highest success rate at 54.67%. This outcome highlights the efficiency of a concise reasoning process, where focused thought generation optimally utilizes limited query resources without diluting the model's attention across less relevant reasoning paths. Notably, the final step's performance at 48% reflects effective synthesis of knowledge accumulated from earlier reasoning stages, facilitated by the Knowledge Propagation Module (KPM). KPM plays a critical role in this process by maintaining and propagating both selected and non-selected thoughts across stages, ensuring that valuable reasoning threads are not prematurely discarded and that accumulated knowledge is effectively leveraged.

$$\text{Success Rate} = \frac{\text{Total Queries Solved}}{\text{Total Queries Attempted}} \times 100\% \tag{13}$$

Increasing the thoughts per step to five resulted in a reduced success rate of 48.00%. This decline is attributed to increased reasoning complexity, which, while offering broader exploration, introduces cognitive noise that can obscure the optimal reasoning path. The final stage under this configuration saw a success rate of 40%, underscoring the diminishing returns of additional reasoning depth when not effectively managed.

Table 3. Step-wise problem-solving performance with varying thoughts per step (3, 5, 7) across Easy, Intermediate, and Final stages. The 3 Thoughts Count/Step configuration yielded the best overall performance, as shown by solved problems, total problems, and success rates.

Thoughts Count/Step	Step	Queries Solved	Queries Attempted	Success Rate (%)	Queries per Solved Problem
3	Easy	8	25	32	3.13
	Intermediate	9	25	36	2.78
	Final	12	25	48	2.08
	Total	**29**	**75**	**54.67↑**	**2.59**
5	Easy	6	25	24	4.17
	Intermediate	7	25	28	3.57
	Final	10	25	40	2.50
	Total	**23**	**75**	**48↓**	**3.26**
7	Easy	4	25	16	6.25
	Intermediate	5	25	20	5.00
	Final	8	25	32	3.13
	Total	**17**	**75**	**38.67↓**	**4.41**

The configuration with seven thoughts per step yielded the lowest success rate at 38.67%. This outcome suggests that excessive reasoning depth overwhelms the model's limited context window, leading to degraded comprehension and decision-making. The final step's performance further dropped to 32%, highlighting the model's struggle to synthesize overly fragmented or diluted reasoning chains. Despite this, KPM mitigates some of these challenges by ensuring that strong and relevant thoughts are propagated, while weak or redundant thoughts are filtered out, preserving reasoning integrity across stages.

Additionally, queries per solved problem metric quantifies the model's efficiency by measuring the average number of queries required to solve a problem, where lower values indicate higher efficiency. This metric highlights how effectively the model utilizes its query budget to achieve correct solutions. The observed values are 2.59, 3.26, and 4.41 for three, five, and seven thoughts per step, respectively, reinforcing the finding that moderate reasoning depth optimizes both performance and resource efficiency.In conclusion, Table 3 demonstrates the critical impact of 'Thoughts Count per Step' on problem-solving performance under a fixed query budget. The results underscore the importance of balancing reasoning depth and efficiency, with KPM serving as a pivotal mechanism for optimizing knowledge propagation and reasoning quality across complex tasks.

5.4 RQ3: Effect of Thought Selection Strategies on Accuracy via KPM

The Knowledge Propagation Module (KPM) in RDoLT provides precise control over the thoughts propagated to subsequent reasoning steps, enabling the selection of the most relevant and high-quality knowledge while discarding less useful information. This selective transmission of knowledge mitigates error accumulation and enhances reasoning efficiency.

Figure 4 illustrates the impact of different thought selection strategies on accuracy. The analysis evaluates four selection approaches: (i) using only selected thoughts, (ii) incorporating both selected and non-selected thoughts, (iii) selecting only the highest-scoring thoughts, and (iv) propagating only the lowest-threshold non-selected thoughts. The results reveal that incorporating both selected and non-selected thoughts results in a wider accuracy range, whereas filtering only final selected thoughts leads to more stable and consistent performance. Specifically, strategies that involve selective knowledge propagation via KPM achieve higher median accuracy, reinforcing the importance of structured filtering mechanisms.

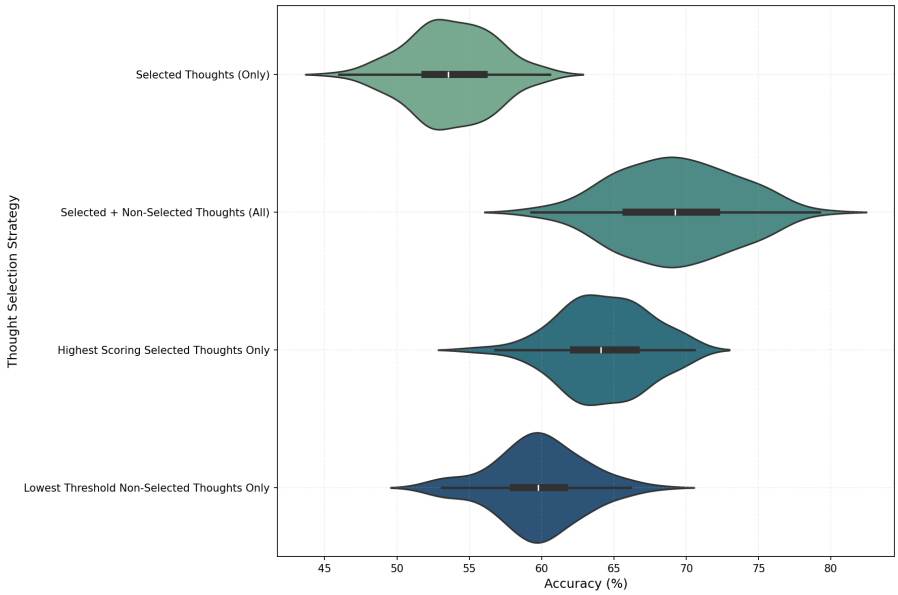

Fig. 4. Impact of different thought selection strategies on accuracy using KPM. Strategies include using only selected thoughts, both selected and non-selected thoughts, highest-scoring selected thoughts, and lowest-threshold non-selected thoughts. Incorporating both selected and non-selected thoughts yields a wider range of accuracy, while selecting only final thoughts ensures more consistent accuracy results

These findings highlight the crucial role of KPM in optimizing thought selection, ensuring that knowledge retained across steps contributes positively to the final reasoning outcome. By refining the propagation of intermediate thoughts, RDoLT enhances prompt efficiency and reasoning consistency, demonstrating that strategic control over knowledge transmission significantly impacts accuracy in multi-step reasoning tasks.

5.5 Performance Analysis and Trade-Offs of Prompting Strategies on GSM8K

Table 4 summarizes the performance of various prompting strategies on GSM8K reasoning tasks, evaluated using the Gemma2-2B [20]. The table reports the average input tokens, output tokens, total tokens per example, inference time, and accuracy. Results suggest that prompts like Vanilla, CoT, CoT-SC, ToT, A-CoT, and L2M—require relatively few tokens (260–957 tokens per example) and have short inference times (1.77–6.46 seconds). However, their accuracy remains modest, with ToT achieving the highest accuracy at 42.8%.

In contrast, RDoLT-based approaches yield substantially higher accuracy. Specifically, the RDoLT method with 3 iterations attains 58.0% accuracy using 3,974 tokens per example and an inference time of 11.94 seconds. The single-iteration RDoLT variant achieves 46.5% accuracy with 1,926 tokens and a 9.22-second inference time, while the unlimited iteration configuration reaches 62.0% accuracy at the cost of 8,564 tokens and a 24.04-second inference time.

These findings reveal a clear trade-off between computational cost and reasoning performance. Although RDoLT-based strategies incur higher token usage and longer processing times, they significantly outperform traditional methods in accuracy. Future research should aim to reduce the token consumption of RDoLT without sacrificing its superior performance, thereby improving its feasibility for practical, real-world applications.

Table 4. Performance Comparison of Different Prompting Strategies on GSM8K Reasoning Tasks (Temperature: 0.4, Threshold: ≥ 30)

Methods	Avg. Input Tokens	Avg. Output Tokens	Avg. Tokens/Example	Time/Example(sec)	Accuracy(%)
Vanilla	76	184	260	1.77 ± 0.5	26.0
CoT	135	299	434	2.71 ± 0.5	38.4
CoT-SC	138	318	456	2.85 ± 0.5	30.5
ToT	274	683	957	6.46 ± 0.5	**42.8**↑
A-CoT	85	185	270	1.81 ± 0.5	34.0
L2M	168	272	440	2.69 ± 0.5	28.5
RDoLT (3 Iterations)	2661	1313	**3974**	11.94 ± 0.5	58.0
RDoLT (1-Iteration)	1033	893	1926	9.22 ± 0.5	46.5
RDoLT (Unlimited)	6676	1888	**8564**↑	24.04 ± 0.5	**62.0**↑

6 Limitations and Future Directions

While the RDoLT framework demonstrates superior performance across multiple benchmarks, several limitations remain. First, the generalizability of RDoLT to domain-specific tasks has not been fully explored. The framework shows promising results in standard reasoning tasks, but its adaptability to highly specialized domains such as legal reasoning or medical diagnostics may require further fine-tuning and optimization. Additionally, the computational overhead of maintaining the Knowledge Propagation Module (KPM) may limit scalability, particularly in resource-constrained environments.

Another potential threat to validity is the reliance on benchmark datasets that may not fully capture the complexity of real-world reasoning scenarios. Although benchmarks like GSM8K and SVAMP are widely used, they represent structured tasks that may not account for the diverse and dynamic nature of human reasoning in more unstructured settings. Another limitation of this study arises from our experimentation with various threshold scoring levels. Determining optimal thresholds proved challenging without conducting an initial evaluation, highlighting a dependency that complicates the setup process.

Future research should focus on addressing these limitations by extending the framework to more domain-specific applications and exploring optimizations that reduce computational costs. Additionally, incorporating more diverse and challenging real-world datasets could provide a deeper evaluation of RDoLT's capabilities. Further work could also investigate hybrid approaches that combine the strengths of multiple prompting strategies to improve performance across various reasoning tasks.

7 Conclusion

This paper introduced the RDoLT framework, a novel approach designed to enhance reasoning in large language models (LLMs) through dynamic thought selection and knowledge propagation. The key innovation, the Knowledge Propagation Module (KPM), ensures that selected and rejected thoughts are tracked and leveraged across reasoning stages, improving accuracy and reducing error propagation. We evaluated RDoLT across multiple benchmarks, including GSM8K, SVAMP, MultiArith, and Gaokao 2023 Math. Our results show that RDoLT consistently outperforms existing methods such as Chain-of-Thought (CoT), CoT with Self-Consistency (CoT-SC), Least-to-Most, and Auto-CoT (A-CoT). For instance, on GSM8K, RDoLT achieved a top accuracy of 90.98% with ChatGPT-4o, surpassing CoT-SC's 89.4%. Similar improvements were observed across LLama 3, Qwen 2, and Gemma 2 models.

What sets RDoLT apart is its ability to utilize rejected thoughts, a feature absent in other methods. This comprehensive approach enhances the reasoning process by maintaining a complete view of generated thoughts, thereby improving overall decision-making. While RDoLT performs exceptionally well on general benchmarks,

further research is needed to optimize its performance for domain-specific tasks and reduce computational overhead. Future work could focus on more efficient knowledge propagation techniques and exploring new domains.

In summary, RDoLT offers a significant advancement in prompt engineering by improving thought selection and knowledge propagation in LLMs. Its performance across diverse benchmarks demonstrates its potential as a flexible and reliable framework for reasoning tasks.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant No. 62471411 and 62071396. Corresponding Author: Zhang Jiashu)

References

[1] M. I. Abdin, S. Gunasekar, V. Chandrasekaran, J. Li, M. Yuksekgonul, R. G. Peshawaria, R. Naik, and B. Nushi. 2023. Kitab: evaluating llms on constraint satisfaction for information retrieval. *arXiv preprint arXiv:2310.15511*.

[2] Q. Ai et al. 2023. Information Retrieval meets Large Language Models: A strategic report from Chinese IR community. *AI Open*, 4, 80–90. DOI: 10.1016/j.aiopen.2023.08.001.

[3] AI@Meta. 2024. Llama 3 model card. https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

[4] Alibaba. 2024. Qwen2 technical report.

[5] T. Alsahfi and K. U. Qasim. 2025. Trafficot-r: a framework for advanced spatio-temporal reasoning in large language models. *Alexandria Engineering Journal*, 128, 464–475. DOI: https://doi.org/10.1016/j.aej.2025.05.027.

[6] Y. Bai et al. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback. (2022). https://arxiv.org/abs/2204.05862 arXiv: 2204.05862 [cs.CL].

[7] M. Besta et al. 2024. Graph of thoughts: solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38, 16, 17682–17690. eprint: 2308.09687. DOI: 10.1609/aaai.v38i16.29720.

[8] M. Besta et al. 2024. Topologies of reasoning: demystifying chains, trees, and graphs of thoughts. *arXiv*. eprint: 2401.14295. DOI: 10.48550/arxiv.2401.14295.

[9] T. Bui, O. Tran, P. Nguyen, B. Ho, L. Nguyen, T. Bui, and T. Quan. 2024. Cross-data knowledge graph construction for llm-enabled educational question-answering system: a case study at hcmut. In *Proceedings of the 1st ACM Workshop on AI-Powered Q&A Systems for Multimedia*, 36–43.

[10] C. Chen, K. Liu, Z. Chen, Y. Gu, Y. Wu, M. Tao, Z. Fu, and J. Ye. 2024. Inside: llms' internal states retain the power of hallucination detection. *arXiv preprint arXiv:2402.03744*.

[11] X. Chen, M. Lin, N. Schärli, and D. Zhou. 2023. Teaching large language models to self-debug. (2023). https://arxiv.org/abs/2304.05128 arXiv: 2304.05128 [cs.CL].

[12] ChilleD. 2023. ChilleD/LastLetterConcat · Datasets at Hugging Face. (2023). Retrieved Sept. 3, 2024 from https://huggingface.co/datasets/ChilleD/LastLetterConcat.

[13] ChilleD. 2023. ChilleD/MultiArith · Datasets at Hugging Face. (2023). Retrieved Sept. 3, 2024 from https://huggingface.co/datasets/ChilleD/MultiArith.

[14] Chilled and Chilled. 2023. Chilled/lastletterconcat · datasets at hugging face. https://huggingface.co/datasets/ChilleD/LastLetterConcat.

[15] K. Cobbe et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

[16] Y. Ding, X. Zhang, S. Amiri, N. Cao, H. Yang, A. Kaminski, C. Esselink, and S. Zhang. 2023. Integrating action knowledge and llms for task planning and situation handling in open worlds. *Autonomous Robots*, 47, 8, 981–997.

[17] A. Elgohary, C. Meek, M. Richardson, A. Fourney, G. Ramos, and A. H. Awadallah. 2021. Nl-edit: correcting semantic parse errors through natural language interaction. (2021). https://arxiv.org/abs/2103.14540 arXiv: 2103.14540 [cs.CL].

[18] E. Erdem et al. 2022. Neural natural language generation: a survey on multilinguality, multimodality, controllability and learning. *Journal of Artificial Intelligence Research*, 73, 1131–1207.

[19] D. Gao et al. 2024. Llms-based machine translation for e-commerce. *Expert Systems with Applications*, 125087.

[20] G. Gemma-Team. 2024. Gemma. DOI: 10.34740/KAGGLE/M/3301.

[21] G. Gendron, Q. Bao, M. Witbrock, and G. Dobbie. 2023. Large language models are not strong abstract reasoners. *arXiv*. eprint: 2305.19555. DOI: 10.48550/arxiv.2305.19555.

[22] A. Gokul. 2023. Llms and ai: understanding its reach and impact.

[23] R. Hong, H. Zhang, X. Pan, D. Yu, and C. Zhang. 2024. Abstraction-of-thought makes language models better reasoners. *arXiv*. eprint: 2406.12442. DOI: 10.48550/arxiv.2406.12442.

[24] J. Huang, X. Chen, S. Mishra, H. S. Zheng, A. W. Yu, X. Song, and D. Zhou. 2023. Large language models cannot self-correct reasoning yet. *arXiv*. eprint: 2310.01798. DOI: 10.48550/arxiv.2310.01798.

[25] T. Khot, H. Trivedi, M. Finlayson, Y. Fu, K. Richardson, P. Clark, and A. Sabharwal. 2023. Decomposed prompting: a modular approach for solving complex tasks. (2023). https://arxiv.org/abs/2210.02406 arXiv: 2210.02406 [cs.CL].

[26] G. Kim, S. Kim, B. Jeon, J. Park, and J. Kang. 2023. Tree of clarifications: answering ambiguous questions with retrieval-augmented large language models. (2023). https://arxiv.org/abs/2310.14696 arXiv: 2310.14696 [cs.CL].

[27] H. Le, Y. Wang, A. D. Gotmare, S. Savarese, and S. C. H. Hoi. 2022. Coderl: mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems*. S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, (Eds.) Vol. 35. Curran Associates, Inc., 21314–21328. https://proceedings.neurips.cc/paper_files/paper/2022/file/8636419dea1aa9fbd25fc4248e702da4-Paper-Conference.pdf.

[28] S. S. Li, V. Balachandran, S. Feng, J. Ilgen, E. Pierson, P. W. Koh, and Y. Tsvetkov. 2024. Mediq: question-asking llms for adaptive and reliable medical reasoning. *arXiv preprint arXiv:2406.00922*.

[29] B. Y. Lin, W. Zhou, M. Shen, P. Zhou, C. Bhagavatula, Y. Choi, and X. Ren. 2020. CommonGen: a constrained text generation challenge for generative commonsense reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. T. Cohn, Y. He, and Y. Liu, (Eds.) Association for Computational Linguistics, Online, (Nov. 2020), 1823–1840. DOI: 10.18653/v1/2020.findings-emnlp.165.

[30] J. Liu, S. Hallinan, X. Lu, P. He, S. Welleck, H. Hajishirzi, and Y. Choi. 2022. Rainier: reinforced knowledge introspector for commonsense question answering. (2022). https://arxiv.org/abs/2210.03078 arXiv: 2210.03078 [cs.CL].

[31] Z. Liu, C. Chen, J. Wang, M. Chen, B. Wu, X. Che, D. Wang, and Q. Wang. 2024. Make llm a testing expert: bringing human-like interaction to mobile gui testing via functionality-aware decisions. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 1–13.

[32] X. Lu, S. Welleck, J. Hessel, L. Jiang, L. Qin, P. West, P. Ammanabrolu, and Y. Choi. 2022. Quark: controllable text generation with reinforced unlearning. In *Advances in Neural Information Processing Systems*. S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, (Eds.) Vol. 35. Curran Associates, Inc., 27591–27609. https://proceedings.neurips.cc/paper_files/paper/2022/file/b125999bde7e80910cbdbd323087df8f-Paper-Conference.pdf.

[33] H. Luo et al. 2024. Wizardmath: empowering mathematical reasoning for large language models via reinforced evol-instruct. (2024). https://arxiv.org/abs/2308.09583 arXiv: 2308.09583 [cs.CL].

[34] A. Madaan et al. 2023. Self-refine: iterative refinement with self-feedback. (2023). https://arxiv.org/abs/2303.17651 arXiv: 2303.17651 [cs.CL].

[35] A. Mahmood, J. Wang, B. Yao, D. Wang, and C.-M. Huang. 2023. Llm-powered conversational voice assistants: interaction patterns, opportunities, challenges, and design guidelines. *arXiv preprint arXiv:2309.13879*.

[36] P. Mirowski, K. W. Mathewson, J. Pittman, and R. Evans. 2022. Co-writing screenplays and theatre scripts with language models: an evaluation by industry professionals. (2022). https://arxiv.org/abs/2209.14958 arXiv: 2209.14958 [cs.HC].

[37] D. Myers et al. 2024. Foundation and large language models: fundamentals, challenges, opportunities, and social impacts. *Cluster Computing*, 27, 1, 1–26.

[38] Ollama. 2024. Ollama: Get up and running with Llama 3.1, Mistral, Gemma 2, and other large language models. (2024). https://github.com/ollama/ollama/tree/main.

[39] OpenAI. 2024. Gpt-4 technical report. (2024). https://arxiv.org/abs/2303.08774 arXiv: 2303.08774 [cs.CL].

[40] A. Patel, S. Bhattamishra, and N. Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online, (June 2021), 2080–2094. DOI: 10.18653/v1/2021.naacl-main.168.

[41] D. Paul, M. Ismayilzada, M. Peyrard, B. Borges, A. Bosselut, R. West, and B. Faltings. 2024. Refiner: reasoning feedback on intermediate representations. (2024). https://arxiv.org/abs/2304.01904 arXiv: 2304.01904 [cs.CL].

[42] N. Pourkamali and S. E. Sharifi. 2024. Machine translation with large language models: prompt engineering for persian, english, and russian directions. https://arxiv.org/abs/2401.08429 arXiv: 2401.08429 [cs.CL].

[43] R. L. Rosa, C. Hulse, and B. Liu. 2024. Can github issues be solved with tree of thoughts? *arXiv*. eprint: 2405.13057. DOI: 10.48550/arxiv.2405.13057.

[44] A. Satpute, N. Gießing, A. Greiner-Petter, M. Schubotz, O. Teschke, A. Aizawa, and B. Gipp. 2024. Can llms master math? investigating large language models on math stack exchange. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR '24). Association for Computing Machinery, Washington DC, USA, 2316–2320. ISBN: 9798400704314. DOI: 10.1145/3626772.3657945.

[45] W. Shen, C. Li, H. Chen, M. Yan, X. Quan, H. Chen, J. Zhang, and F. Huang. 2024. Small llms are weak tool learners: a multi-llm agent. *arXiv*. eprint: 2401.07324. DOI: 10.48550/arxiv.2401.07324.

[46] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao. 2023. Reflexion: language agents with verbal reinforcement learning. (2023). https://arxiv.org/abs/2303.11366 arXiv: 2303.11366 [cs.AI].

[47] K. Stechly, K. Valmeekam, and S. Kambhampati. 2024. Chain of thoughtlessness? an analysis of cot in planning. (2024). https://arxiv.org/abs/2405.04776 arXiv: 2405.04776 [cs.AI].

[48] N. Tandon, A. Madaan, P. Clark, and Y. Yang. 2021. Learning to Repair: Repairing model output errors after deployment using a dynamic memory of feedback. *arXiv*. eprint: 2112.09737. DOI: 10.48550/arxiv.2112.09737.

[49] R. Tseng, S. Verberne, and P. van der Putten. 2023. Chatgpt as a commenter to the news: can llms generate human-like opinions? In *Multidisciplinary International Symposium on Disinformation in Open Online Media*. Springer, 160–174.

[50] M. Van Der Meer, E. Liscio, C. Jonker, A. Plaat, P. Vossen, and P. Murukannaiah. 2024. A hybrid intelligence method for argument mining. *Journal of Artificial Intelligence Research*, 80, 1187–1222.

[51] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv*. eprint: 2203.11171. DOI: 10.48550/arxiv.2203.11171.

[52] Z. Wang, S. Cai, G. Chen, A. Liu, X. Ma, and Y. Liang. 2024. Describe, explain, plan and select: interactive planning with large language models enables open-world multi-task agents. (2024). https://arxiv.org/abs/2302.01560 arXiv: 2302.01560 [cs.AI].

[53] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. *arXiv*. eprint: 2201.11903. DOI: 10.48550/arxiv.2201.11903.

[54] S. Welleck, X. Lu, P. West, F. Brahman, T. Shen, D. Khashabi, and Y. Choi. 2022. Generating sequences by learning to self-correct. (2022). https://arxiv.org/abs/2211.00053 arXiv: 2211.00053 [cs.CL].

[55] Y. Wu, J. Zhang, N. Hu, L. Tang, G. Qi, J. Shao, J. Ren, and W. Song. 2024. Mldt: multi-level decomposition for complex long-horizon robotic task planning with open-source large language model. (2024). https://arxiv.org/abs/2403.18760 arXiv: 2403.18760 [cs.RO].

[56] Z. Xi, S. Jin, Y. Zhou, R. Zheng, S. Gao, T. Gui, Q. Zhang, and X. Huang. 2024. Self-polish: enhance reasoning in large language models via problem refinement. (2024). https://arxiv.org/abs/2305.14497 arXiv: 2305.14497 [cs.CL].

[57] Y. Xie, K. Kawaguchi, Y. Zhao, X. Zhao, M.-Y. Kan, J. He, and Q. Xie. 2023. Self-evaluation guided beam search for reasoning. (2023). https://arxiv.org/abs/2305.00633 arXiv: 2305.00633 [cs.CL].

[58] F. Xing. 2024. Designing heterogeneous llm agents for financial sentiment analysis. *ACM Transactions on Management Information Systems*.

[59] S. Xu, W. Xie, L. Zhao, and P. He. 2025. Chain of draft: thinking faster by writing less. (2025). https://arxiv.org/abs/2502.18600 arXiv: 2502.18600 [cs.CL].

[60] Q. Xue. 2024. Unlocking the potential: A comprehensive exploration of large language models in natural language processing. *Applied and Computational Engineering*, 57, 1, 247–252. DOI: 10.54254/2755-2721/57/20241341.

[61] A. B. Yadav. 2024. Generative ai in the era of transformers: revolutionizing natural language processing with llms. *Journal of Image Processing and Intelligent Remote Sensing*, 42, 54–61. DOI: 10.55529/jipirs.42.54.61.

[62] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. 2023. Tree of thoughts: deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36. https://arxiv.org/abs/2305.10601v2.

[63] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. 2023. React: synergizing reasoning and acting in language models. (2023). https://arxiv.org/abs/2210.03629 arXiv: 2210.03629 [cs.CL].

[64] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. 2023. React: synergizing reasoning and acting in language models. (2023). https://arxiv.org/abs/2210.03629 arXiv: 2210.03629 [cs.CL].

[65] P. Yu, H. Xu, X. Hu, and C. Deng. 2023. Leveraging generative ai and large language models: a comprehensive roadmap for healthcare integration. In *Healthcare* number 20. Vol. 11. MDPI, 2776.

[66] H. Zhang, P. S. Yu, and J. Zhang. 2024. A systematic survey of text summarization: from statistical methods to large language models. *arXiv preprint arXiv:2406.11289*.

[67] J. Zhang, X. Xu, and S. Deng. 2023. Exploring collaboration mechanisms for llm agents: a social psychology view. *arXiv preprint arXiv:2310.02124*.

[68] X. Zhang, C. Li, Y. Zong, Z. Ying, L. He, and X. Qiu. 2023. Evaluating the performance of large language models on gaokao benchmark. In.

[69] Z. Zhang, A. Zhang, M. Li, and A. Smola. 2022. Automatic chain of thought prompting in large language models. *arXiv*. eprint: 2210.03493. DOI: 10.48550/arxiv.2210.03493.

[70] L. Zheng et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. (2023). https://arxiv.org/abs/2306.05685 arXiv: 2306.05685 [cs.CL].

[71] B. Zhou, D. Geißler, and P. Lukowicz. 2024. Misinforming llms: vulnerabilities, challenges and opportunities. (2024). https://arxiv.org/abs/2408.01168 arXiv: 2408.01168 [cs.CL].

[72] D. Zhou et al. 2022. Least-to-most prompting enables complex reasoning in large language models. *The International Conference on Learning Representations (ICLR)*. http://arxiv.org/abs/2205.10625.

A Variants of Reasoning Strategies in RDoLT

To ensure clarity in our experimental setup, we describe the different reasoning strategies (variants) evaluated in this study, as mentioned in Table 2. These approaches vary based on the number of requests made to the model and the level of decomposition applied during reasoning.

A.1 Single-Step (Sequential) / Single Request

This approach completes the entire reasoning process of RDoLT in a single request to the LLM. The model processes the input sequentially, generating a response in one go without revisiting or refining thoughts beyond the initial attempt.

A.2 Single-Step (One-Shot) / Single Request

Similar to the sequential approach, this method consists of a single request. However, the model is prompted to generate the complete reasoning chain in a single inference step, with one full example of the entire process provided.

A.3 Few-Shot (Two Examples)

In this variant, the model is provided with two demonstration examples before generating a response. These examples serve as guidance, allowing the model to learn from structured reasoning before attempting the given task. Few-shot prompting has been widely studied as a means to improve reasoning performance in LLMs without requiring fine-tuning.

A.4 Multi-Requests (Maximum 3 Iterations)

This strategy allows the model to make up to three requests to refine its reasoning iteratively. If a generated response falls below the threshold, the model can regenerate thoughts or adjust reasoning up to three times before selecting the best-performing response based on predefined scoring criteria.

A.5 Multi-Requests (Unlimited Iterations)

Unlike the three-iteration limit, this approach allows the model to generate and refine thoughts indefinitely until an optimal solution is found. If responses fail the scoring process, additional reasoning steps are triggered dynamically. While this method can improve accuracy, it is significantly more computationally expensive, as mentioned in Table 4, requiring higher memory and longer processing times.

B Benchmark Task Walkthroughs

Generally, RDoLT requires only three requests and corresponding prompts: Task Decomposition, Thought Generation, and Thoughts Evaluation. In the following section, we present some results that demonstrate the generalization of RDoLT across a variety of benchmarks.

B.1 Last Letter Concatenation

```
Task: "Take the last letter of each word in the sentence: 'Artificial intelligence is the
    future' and concatenate them to form a new string."
```

```
Decomposition [Easy]
Description: Extract the last letter of each word in the sentence.
```

```
Thought 1: Identify the words in the sentence: "Artificial", "intelligence", "is", "the", "
    future".
Thought 2: Extract the last letter of each word: "l", "e", "s", "e", "e".
Thought 3: Check if each word is correctly identified and its last letter is accurately
    extracted.

Scores:
- Thought 1: LV: 10, COH: 9, SIM: 10, ADP: 10, Total: 39
- Thought 2: LV: 9, COH: 8, SIM: 8, ADP: 9, Total: 34
- Thought 3: LV: 5, COH: 5, SIM: 5, ADP: 5, Total: 20
```

```
KPM [Easy Step]
Selected Thoughts (Easy): Thought 1, Thought 2
Rejected Thoughts (Easy): Thought 3
```

```
Decomposition [Intermediate]
Description: Combine the extracted letters into a single string.

Thought 1: Concatenate the letters to form the string "lesee".
Thought 2: Verify if the concatenated string represents the order of the words in the original
     sentence.
Thought 3: Check for any errors in the concatenation process.

Scores:
- Thought 1: LV: 10, COH: 9, SIM: 9, ADP: 10, Total: 38
- Thought 2: LV: 8, COH: 7, SIM: 8, ADP: 8, Total: 31
- Thought 3: LV: 4, COH: 5, SIM: 4, ADP: 5, Total: 18
```

```
KPM [Intermediate Step]
Selected Thoughts (Easy): Thought 1, Thought 2
Rejected Thoughts (Easy): Thought 3
Selected Thoughts (Intermediate): Thought 1, Thought 2
Rejected Thoughts (Intermediate): Thought 3
```

```
Decomposition [Final]
Description: Verify the final output and ensure all steps have been completed correctly.

Thought 1: Review the concatenated string "lesee" to ensure it matches the task requirements.
Thought 2: Validate that the process of extracting last letters and concatenating them
    followed the proper sequence.
Thought 3: Finalize the result as the correct output for the given task.

Scores:
- Thought 1: LV: 10, COH: 10, SIM: 9, ADP: 10, Total: 39
- Thought 2: LV: 7, COH: 7, SIM: 7, ADP: 7, Total: 28
- Thought 3: LV: 4, COH: 4, SIM: 4, ADP: 4, Total: 16

Final Answer: The final concatenated string is "lesee".
```

```
KPM [Final Step]
Selected Thoughts (Easy): Thought 1, Thought 2
Rejected Thoughts (Easy): Thought 3
Selected Thoughts (Intermediate): Thought 1, Thought 2
```

```
Rejected Thoughts (Intermediate): Thought 3
Selected Thoughts (Final): Thought 1
Rejected Thoughts (Final): Thought 2, Thought 3

Final Answer: "lesee"
```

B.2 GSM8K

```
Task: "Toula went to the bakery and bought various types of pastries. She bought 3 dozen
    donuts which cost $68 per dozen, 2 dozen mini cupcakes which cost $80 per dozen, and 6
    dozen mini cheesecakes for $55 per dozen. How much was the total cost?"
```

```
Decomposition [Easy]
Description: Break down the task by identifying each type of pastry and calculating the total
    cost for each one individually.

Thought 1: Identify the total cost of the donuts by multiplying 3 dozen by $68 per dozen.
Thought 2: Identify the total cost of the mini cupcakes by multiplying 2 dozen by $80 per
    dozen.
Thought 3: Identify the total cost of the mini cheesecakes by multiplying 6 dozen by $55 per
    dozen.

Scores:
- Thought 1: LV: 10, COH: 9, SIM: 10, ADP: 10, Total: 39
- Thought 2: LV: 9, COH: 9, SIM: 9, ADP: 9, Total: 36
- Thought 3: LV: 6, COH: 6, SIM: 6, ADP: 7, Total: 25
```

```
KPM [Easy Step]
Selected Thoughts (Easy): Thought 1, Thought 2
Rejected Thoughts (Easy): Thought 3
```

```
Decomposition [Intermediate]
Description: Calculate the cost for each pastry and find the sum of all the items.

Thought 1: The total cost of the donuts is 3 * 68 = 204.
Thought 2: The total cost of the mini cupcakes is 2 * 80 = 160.
Thought 3: The total cost of the mini cheesecakes is 6 * 55 = 330.

Scores:
- Thought 1: LV: 10, COH: 10, SIM: 9, ADP: 10, Total: 39
- Thought 2: LV: 9, COH: 9, SIM: 9, ADP: 9, Total: 36
- Thought 3: LV: 6, COH: 7, SIM: 7, ADP: 7, Total: 27
```

```
KPM [Intermediate Step]
Selected Thoughts (Easy): Thought 1, Thought 2
Rejected Thoughts (Easy): Thought 3
Selected Thoughts (Intermediate): Thought 1, Thought 2
Rejected Thoughts (Intermediate): Thought 3
```

```
Decomposition [Final]
Description: Calculate the total cost by adding up the costs from the previous steps.
```

```
Thought 1: Add the total cost of the donuts, mini cupcakes, and mini cheesecakes to get the
    final total: 204 + 160 + 330.
Thought 2: Verify that the calculation for each item was done correctly.
Thought 3: Double-check the total to make sure there were no mistakes in the addition.

Scores:
- Thought 1: LV: 10, COH: 10, SIM: 10, ADP: 10, Total: 40
- Thought 2: LV: 9, COH: 9, SIM: 9, ADP: 9, Total: 36
- Thought 3: LV: 7, COH: 7, SIM: 7, ADP: 7, Total: 28

Final Answer: The total cost is $694.
```

```
KPM [Final Step]
Selected Thoughts (Easy): Thought 1, Thought 2
Rejected Thoughts (Easy): Thought 3
Selected Thoughts (Intermediate): Thought 1, Thought 2
Rejected Thoughts (Intermediate): Thought 3
Selected Thoughts (Final): Thought 1, Thought 2
Rejected Thoughts (Final): Thought 3

Final Answer: $694
```

B.3 MMLU

```
Task: "Let p = (1, 2, 5, 4)(2, 3) in S_5. Find the index of <p> in S_5. Choices: ['8', '2',
    '24', '120']"
```

```
Decomposition [Easy]
Description: Break down the task by identifying and analyzing the given permutation structure
    in S_5.

Thought 1: Recognize that p is a product of disjoint cycles, where (1, 2, 5, 4) is a 4-cycle
    and (2, 3) is a 2-cycle.
Thought 2: Identify that the order of a permutation is the least common multiple (LCM) of the
    lengths of the disjoint cycles.
Thought 3: Calculate the order of p using the LCM of 4 and 2.

Scores:
- Thought 1: LV: 10, COH: 10, SIM: 10, ADP: 9, Total: 39
- Thought 2: LV: 9, COH: 9, SIM: 9, ADP: 8, Total: 35
- Thought 3: LV: 6, COH: 7, SIM: 7, ADP: 6, Total: 26
```

```
KPM [Easy Step]
Selected Thoughts (Easy): Thought 1, Thought 2
Rejected Thoughts (Easy): Thought 3
```

```
Decomposition [Intermediate]
Description: Analyze the structure of the group <p> and its implications for the index in S_5.

Thought 1: The subgroup <p> is generated by p, and the order of p is the least common multiple
     (LCM) of 4 and 2, which is 4.
Thought 2: Calculate the index of <p> in S_5 as 120 / 4 = 30 (incorrect thought).
```

```
Thought 3: Recognize that the correct formula for the index of a cyclic subgroup in a
    symmetric group is 120 / 4 = 30.

Scores:
- Thought 1: LV: 10, COH: 10, SIM: 9, ADP: 9, Total: 38
- Thought 2: LV: 5, COH: 6, SIM: 5, ADP: 5, Total: 21
- Thought 3: LV: 9, COH: 9, SIM: 9, ADP: 9, Total: 36
```

```
KPM [Intermediate Step]
Selected Thoughts (Easy): Thought 1, Thought 2
Rejected Thoughts (Easy): Thought 3
Selected Thoughts (Intermediate): Thought 1, Thought 3
Rejected Thoughts (Intermediate): Thought 2
```

```
Decomposition [Final]
Description: Use all previous knowledge to determine the correct index of <p> in S_5.

Thought 1: The order of S_5 is 120, and the order of p is 4, so the index of <p> in S_5 is 30.
Thought 2: Verify if the solution aligns with the structure of S_5 and correct subgroup orders
    .
Thought 3: Conclude that the index of <p> in S_5 is 2 based on the correct analysis of p's
    structure.

Scores:
- Thought 1: LV: 7, COH: 8, SIM: 7, ADP: 7, Total: 29
- Thought 2: LV: 9, COH: 9, SIM: 9, ADP: 9, Total: 36
- Thought 3: LV: 10, COH: 10, SIM: 9, ADP: 10, Total: 39

Final Answer: The index of <p> in S_5 is 2.
```

```
KPM [Final Step]
Selected Thoughts (Easy): Thought 1, Thought 2
Rejected Thoughts (Easy): Thought 3
Selected Thoughts (Intermediate): Thought 1, Thought 3
Rejected Thoughts (Intermediate): Thought 2
Selected Thoughts (Final): Thought 2, Thought 3
Rejected Thoughts (Final): Thought 1

Final Answer: 2
```

C Prompt Structure Illustrations

Table 5. A Feature-Based Comparison of RDoLT and Experimental Baselines

Key Architectural Feature	RDoLT	Vanilla	CoT	CoT-SC	L2M	A-CoT	Self-Polish	CoD	ReAct
1. Decomposes Problem *Breaks task into smaller steps*	✓ (Hierarchical Tiers)	✗	✓ (Implicit Steps)	✓ (Implicit Steps)	✓ (Sequential Sub-problems)	✓ (Implicit Steps)	✓ (Via Draft & Refine)	✓ (Via Draft & Refine)	✓ (Thought-Act Cycle)
2. Explores Multiple Paths *Avoids committing to one path*	✓	✗	✗	✓	✗	✗	✗	✗	✗
3. Iterative Refinement *Improves upon an initial draft*	✓ (Via structured tiers)	✗	✗	✗	✗	✗	✓	✓	✗
4. Explicit Intermediate Evaluation *Checks quality of steps before the end*	✓ (Multi-feature scoring)	✗	✗	✗ (Votes on final answer only)	✗	✗	△ (Via self-critique)	△ (Via self-critique)	△ (Via tool output)
5. Learns from Rejected Thoughts *Uses mistakes to guide reasoning*	✓ **(Unique Feature)**	✗	✗	✗	✗	✗	✗	✗	✗
6. Uses External Tools *Can act on the environment*	✗	✗	✗	✗	✗	✗	✗	✗	✓

```
Let's solve the following problem using the Tree-of-Thought approach.

Question: {task}

Step 1: Generate Initial Thoughts (Breadth = 3)
-------------------------------------------------
Provide three distinct initial approaches to tackle the problem:
   - Thought A: [Describe initial approach 1]
   - Thought B: [Describe initial approach 2]
   - Thought C: [Describe initial approach 3]

Step 2: Deepen a Promising Thought (Depth = 2)
-----------------------------------------------
From the initial thoughts, select the most promising one based on preliminary evaluation.
For the selected thought, elaborate with two levels of sub-thoughts:
   Selected Thought: [Indicate which thought is selected]
      ├─ Sub-thought 1: [Detailed reasoning step]
      ├─ Sub-thought 2: [Detailed reasoning step]
      └─ Sub-thought 3: [Detailed reasoning step]

Step 3: Evaluate the Reasoning Branches
---------------------------------------
For each branch generated in Step 2, please:
   - Rate the likelihood of success (scale: 1 to 10)
   - Assess computational feasibility
   - Identify potential dead ends or weaknesses

Step 4: Converge on the Best Path and Conclude
-----------------------------------------------
Based on the evaluations, select the best thought chain.
   Best Thought Chain: [Present the selected chain with supporting reasoning]
      └─ Final Detailed Solution: [Elaborate the final solution based on the best chain]

Final Answer: [Provide the final answer here]
```

Fig. 5. Tree-of-Thought Prompt illustrating a structured problem-solving framework that employs a breadth of 3 initial thoughts and a depth of 2 levels of sub-thoughts to systematically explore and evaluate multiple reasoning paths, ultimately converging on the optimal solution.

```
### Step 1: Decomposition
Decompose the complex reasoning task into three sub-tasks: Easy, Intermediate, and Final, each with increasing
complexity.
- **Easy sub-task**: Identify the simplest part of the problem that can be solved or understood with minimal effort.
- **Intermediate sub-task**: Build upon the easy sub-task, using its results to tackle a more complex aspect.
- **Final sub-task**: Combine the solutions from both the easy and intermediate sub-tasks to solve the entire problem.
Ensure each sub-task is logically progressive with a feedback mechanism where each level informs the next.
```

```
# Improved Step 2: Thought Generation
### Step 2: Thought Generation
For each sub-task (Easy, Intermediate, Final), generate {n} candidate thoughts that outline potential steps or solutions.
- For the Intermediate sub-task, build upon the candidate thoughts generated at the Easy level.
- For the Final sub-task, integrate insights from both the Easy and Intermediate levels.
Ensure that all candidate thoughts are diverse, comprehensive, and explore a broad range of reasoning pathways.
```

```
  ### Step 3: Evaluation
  Evaluate each thought based on four core features: **Logical Validity**, **Coherence**, **Simplicity**, and
  **Adaptiveness**. Each feature will be assessed on a scale of 1 to 10, where 1 indicates a weak thought and 10
  represents a strong one. The final score for each thought will be the sum of these four features.

  1. **Logical Validity**: The thought must follow sound logical principles without contradictions or fallacies. A
  penalty is applied for violations of relevant logical rules, such as those governing mathematical operations or
  reasoning principles. The more logical the thought, the higher the score. For instance, a thought that adheres to
  logical rules will score high, while a thought that violates principles such as the order of operations or basic
  reasoning will be penalized.

  2. **Coherence**: This measures how well the thought fits with the previous thoughts in the chain. A thought that
  aligns well with earlier ideas and adds clarity will score higher. The more consistent the thought is with the rest
  of the reasoning process, the more coherent it is. Thoughts that appear disconnected or out of place should receive
  a lower score.

  3. **Simplicity**: Thoughts should be clear and concise. A thought that is overly complex or difficult to understand
  due to verbosity or unnecessary details will score lower. A simpler, more direct thought will receive a higher
  score. The length and depth of a thought, in terms of both its wording and logical steps involved, will influence
  this score. Shorter, direct thoughts with fewer logical steps will score better.

  4. **Adaptiveness**: This feature measures how well the thought aligns with the specific task and query. A thought
  that is highly relevant to the problem, following the guidelines and addressing the task directly, will score
  highly. Thoughts that stray from the context of the task or are less relevant to the query will score poorly.
```

```
### Step 4: Selection
A thought will be selected if its total score is above the predefined threshold (in this case, {threshold}). If no
thought meets the threshold, the system will regenerate thoughts and evaluate them again. This ensures that only high-
quality thoughts are retained at each stage of the reasoning process.
```

```
### Step 5: Knowledge Propagation Module (KPM)
Track both {selected} and {non-selected} thoughts, making them available for the next level. If no thoughts are selected
at any level, the system will regenerate thoughts for further evaluation. Additionally, edge cases where multiple
thoughts score the same or no thoughts meet the threshold will trigger feedback loops to refine the reasoning process.
```

```
### Step 6: Final Answer
Once the thoughts have been evaluated and selected, the final answer will be derived primarily from the {selected}
thoughts at the Final stage. However, the system should also reference any {non-selected} thoughts to ensure that
rejected or suboptimal reasoning is not overlooked. This approach helps confirm the validity of the final solution,
leveraging both selected and non-selected thoughts to produce a robust and contextually aligned answer.

Now, apply this framework to the problem: "{problem}"
```

Fig. 6. Illustration of the RDoLT single-request, zero-shot prompt highlighting its core components.

Received 10 March 2025; accepted 29 June 2025.