Divide, Verify, and Conquer: Verification-Guided DAG Reasoning for Large Language Models

Yigeng Jiang*, Tingjun Su*, Tong Wu†, Shumeng Zhang‡, Yanxu Zhao§, Zhaohong Huang*,
Tianyu Xie*, Yuhang Wu*, Yisheng Lin*, Yuze Liu¶, Fei Chao*, Xiawu Zheng*‖
*Key Laboratory of Multimedia Trusted Perception and Efficient Computing,
Ministry of Education of China, Xiamen University, 361005, P.R. China
†Du Xiaoman Finance, Beijing, China
‡School of Informatics, Xiamen University, Xiamen, China
§School of Mathematical Sciences, Xiamen University, Xiamen, China
¶Shanghai AI Lab
*Email: {36920231153200, 37220232203813, zhaohonghuang}@stu.xmu.edu.cn,
alexisty233@gmail.com, {derekwu2020, 19820231154000}@stu.xmu.edu.cn,
{fchao, zhengxiawu}@xmu.edu.cn,
†Email: tonwuton@gmail.com
‡Email: shumz@stu.xmu.edu.cn
§Email: 19020232202447@stu.xmu.edu.cn
¶Email: yuzeliu@swin.edu.au
‖Corresponding author: zhengxiawu@xmu.edu.cn

***Abstract*—Chain-of-Thought (CoT) prompting is widely adopted for mathematical and scientific reasoning by decomposing problems into stepwise chains. However, the lack of intermediate verification leads to error propagation, where early mistakes can invalidate the entire reasoning process. To address this limitation, we introduce Divide, Verify, and Conquer (DVC), a training-free framework that models problem-solving as a DAG rather than a linear chain. DVC employs meta-rules to guide two key transformations: Problem Equivalence (PE) for logically equivalent reformulations and Problem Decomposition (PD) for breaking problems into manage-able subproblems with explicit dependency modeling. We design dedicated verification templates combined with a multi-round reflection mechanism to ensure each transformation step is reliable, effectively reducing error accumulation. Extensive experiments conducted on challenging mathematical and scientific reasoning benchmarks clearly demonstrate the effectiveness of the proposed DVC framework, outperforming existing methods with improved accuracy and reliability.**

***Index Terms*—Large language models, Multi-step Reasoning, In-context Learning.**

I. Introduction

Mathematical and scientific reasoning constitute fundamental pillars in the pursuit of artificial intelligence, demanding sophisticated logical deduction, multi-step problem decomposition, and systematic thinking [1]–[4]. The ability to tackle complex mathematical and scientific problems has become a crucial benchmark for evaluating progress toward higher level intelligence in large language models (LLMs) [5], [6].Recently, reinforcement learning (RL) has emerged as a prominent paradigm for enhancing the reasoning capabilities of LLMs [7]–[9]. However, these approaches typically demand substantial computational resources and intensive training procedures, which significantly limits their accessibility and scalability in practice [10]–[14]. As an alternative, test-time compute methods have gained attention for their training-free nature, primarily building upon chain-of-thought (CoT) prompting [15] and its variants [16]–[18].

While CoT-based methods offer computational efficiency advantages, they suffer from fundamental limitations that raise serious concerns regarding their reliability and robustness. First, CoT's inherently linear, monolithic chain structure renders it vulnerable to error propagation—a single mistake early in the reasoning chain can cascade through and invalidate all subsequent steps [19], [20]. Second, intermediate reasoning steps lack explicit verification mechanisms, undermining interpretability and providing weak guarantees about logical soundness or completeness [21], [22]. Third, when problems require decomposition into subproblems, dependencies among these components are handled implicitly, often leading to inefficient or incorrect reasoning in scenarios with complex interrelations [19], [23]. These limitations challenge the foundational assumptions underlying current CoT-based approaches and highlight the urgent need for more principled frameworks for complex problem solving.

To address these challenge, we argue that effective complex reasoning should not merely unfold as a sequential chain of steps, but rather should be structured to enable explicit correctness verification and controlled error propagation. Our key insight is that any complex reasoning task can be systematically modeled through two fundamental operations: **(1) PE** — transforming problems into logically equivalent but more tractable forms, and **(2) PD** — breaking complex problems into simpler subproblems. This decomposition strategy enables

systematic exploration of the solution space while maintaining logical soundness through explicit verification at each transformation step.

Building upon this insight, we present **DVC**, a novel training-free reasoning framework that structures complex problem solving as the dynamic construction of a **directed acyclic graph (DAG)** of subproblems and transformations. First, DVC employs **meta-rules**—implemented via structured prompts with explicit constraints and few-shot examples—to systematically govern PE and PD transformations. Additionally, each transformation is validated by a dedicated **Verifier** module that ensures logical equivalence for PE operations and completeness for PD decompositions. Most importantly, when verification fails, DVC utilizes a **multi-round reflection** mechanism that generates structured feedback, revises transformations under meta-rule guidance, and iteratively refines operations until validity is achieved. An **Aggregator** module then synthesizes the final solution by traversing the verified DAG in topological order.

Our contributions are as follows: (1) We introduce DVC, a novel training-free framework that structures reasoning as a verifiable DAG-based process, fundamentally departing from linear chain approaches. (2) We design principled meta-rules and structured prompting strategies with dedicated verifier and aggregator modules that enable rigorous verification and multi-round reflection. (3) We provide comprehensive experimental validation showing DVC significantly outperforms CoT methods across challenging mathematical and scientific reasoning benchmarks. Our code is available at https://github.com/yigengjiang/Divide-Verify-Conquer.

II. Related Work

A. CoT Prompting

CoT prompting is a widely used technique for eliciting multi-step reasoning in large language models. Wei et al. [15] showed that providing intermediate reasoning steps can improve performance on arithmetic, commonsense, and symbolic reasoning tasks. Building on this line of work, Self-Consistency [16] improves robustness by sampling multiple reasoning trajectories and selecting the final answer via majority voting. Liu [24] studies demonstration selection in the Tree-of-Thoughts framework and proposes TOT-DS to select informative exemplars for reasoning. Nevertheless, linear CoT pipelines remain vulnerable to error propagation, since early mistakes may cascade and invalidate subsequent steps.

B. Structured Reasoning Frameworks

To address the limitations of linear reasoning chains, prior work has explored structured formulations that organize reasoning as trees or graphs. Tree of Thoughts (ToT) [17] performs deliberate search over multiple reasoning branches with lookahead and backtracking. Graph of Thoughts (GoT) [18] further generalizes this idea by representing intermediate thoughts as a graph and enabling flexible aggregation. Zhu [25] proposes SRF, which introduces structured decomposition strategies to enhance LLM reasoning. Huang et al. [26] study graph reasoning in cross-modal settings and propose Kolmogorov–Arnold Network-enhanced alignment for textual attribute graph reasoning. While these structured approaches provide more flexible reasoning topologies than linear chains,they do not always incorporate explicit verification to guarantee the correctness of intermediate transformations.

C. Verification in Reasoning

Recent work highlights the importance of verification for reliable LLM reasoning. Lightman et al. [27] show that process supervision, which provides feedback on intermediate reasoning steps, can outperform outcome-only supervision when training reliable reasoners. Ling et al. [21] propose deductive verification to validate chain-of-thought reasoning, underscoring the need for stronger correctness guarantees. Zheng et al. [28] introduce Critic-CoT, which performs step-wise critique to improve reasoning quality without additional human annotation. Chowdhury and Caragea [29] study zero shot verification-guided reasoning, suggesting that LLMs can verify intermediate steps without fine-tuned verifiers or few shot exemplars. Building on these insights, DVC introduces verification at each transformation step and uses multi-round reflection to improve logical soundness throughout the reasoning process.

III. Method

DVC is designed to solve complex problems by dynamically constructing and resolving a solution DAG. The architecture of our proposed framework is illustrated in Fig 1. The process comprises three core stages: Problem Structuring, Iterative DAG Construction, and Hierarchical Solution Aggregation.

A. Problem Structuring

Most scientific problems contain implicit constraints, redundant irrelevant contexts, and other complexities. Therefore, large language models need to extract key information and filter out irrelevant contexts during the reasoning phase.

Accordingly, we introduce the **Problem Structurer** module before formal reasoning to achieve problem structuring, thereby separating problem extraction from the final reasoning stage. Specifically, we design a meta-prompt ϕ for extracting and structuring problems. The structured problem can be represented as:

$$C, O = \mathrm{LLM}(\phi(\text{task})), \tag{1}$$

where $C = \{c_1, c_2, \ldots, c_k\}$ represents the constraint conditions of the problem, $O = \{o_1, o_2, \ldots, o_m\}$ represents the solution objectives of the problem, and task denotes the original problem.

This structured representation eliminates ambiguity and provides a clear, machine-readable format that serves as the initial state (the root node of the solution DAG) for the subsequent reasoning process. This ensures that all modules operate on a consistent and well-defined problem space.

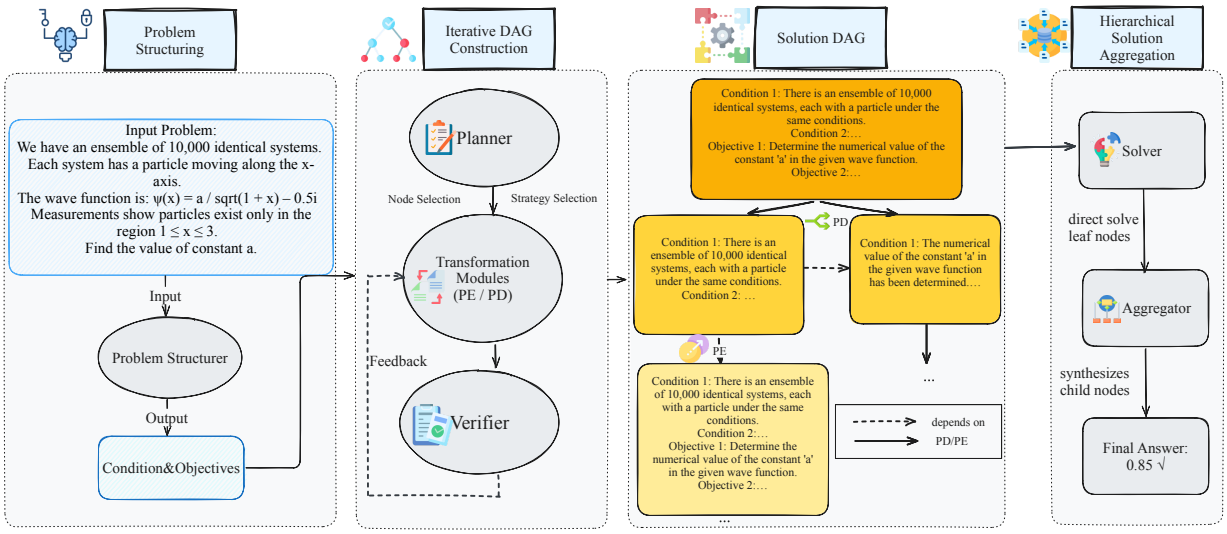

Fig. 1. An illustration of our Divide–Verify–Conquer (DVC) framework on a sample question.

B. Iterative DAG Construction

The core of DVC is an iterative process of building a solution DAG through coordinated transformation and verification operations.

Planner serves as the central coordinator responsible for two critical decisions: node selection and transformation strategy selection.

For node selection, we employ a heuristic that prioritizes: (i) nodes with zero in-degree (no dependencies), (ii) nodes at shallower depths to prevent excessive decomposition, and (iii) nodes for which not all previous transformation attempts have been unsuccessful. The selection function can be formalized as:

$$n^* = \arg\min_{n \in F} \{d(n) : \text{indegree}(n) = 0 \wedge \neg\text{failed}(n)\}, \quad (2)$$

where F represents the frontier nodes (unsolved leaf nodes), $d(n)$ denotes the depth of node n, and $\text{failed}(n)$ indicates whether all transformations for node n have failed.

For strategy selection, the Planner selects the transformation policy $\pi \in \{\text{PE}, \text{PD}\}$ based on the node's state:

$$\pi = \text{LLM}(\phi_{\text{policy}}(n)), \quad (3)$$

where ϕ_{policy} is the policy selection prompt template.

Transformation Modules (PE/PD) are invoked based on the Planner's policy to execute the corresponding transformations.

For Problem Equivalence (PE), we transform problems into logically equivalent but simpler forms:

$$\mathcal{T}_{\text{PE}} : (C, O) \mapsto (C', O'), \quad (4)$$

where the transformation must preserve solution space, ensuring that the two formulations admit the same set of solutions.

For Problem Decomposition (PD), we break down problems into multiple independent sub-problems:

$$\mathcal{T}_{\text{PD}} : (C, O) \mapsto \{(C_1, O_1), (C_2, O_2), \ldots, (C_k, O_k)\}, \quad (5)$$

Each sub-problem (C_i, O_i) is associated with dependency constraints $D_i \subseteq \{1, 2, \ldots, k\} \setminus \{i\}$, creating a directed acyclic graph (DAG) structure where $i \to j$ if $i \in D_j$.

Verifier validates every transformation generated by the PE/PD modules. For PE transformations, it verifies: (1) bidirectional solution equivalence, (2) preservation of domain constraints, and (3) logical consistency. For PD transformations, it checks: (1) **completeness of problem coverage**, (2) **self-containment of each subproblem**, (3) **correctness of dependency annotations**, and (4) **adherence to domain-specific principles**. The verification process employs domain-specific templates and multi-temperature retry mechanisms. When verification fails, the Verifier generates detailed feedback to guide subsequent transformation attempts. This rigorous multi-level verification ensures the logical soundness of each transformation.

C. Hierarchical Solution Aggregation

After the DAG construction, we perform bottom-up solution aggregation.

Solver is responsible for solving the terminal leaf nodes of the DAG. These nodes represent the simplest form of the original problem. And produce its solution S_n as:

$$\text{S}_\text{n} = \text{LLM}_{\text{solve}}(n). \quad (6)$$

where $\text{LLM}_{\text{solve}}$ denotes the instantiated solver with a LLM.

Aggregator recursively synthesizes the solutions from child nodes to derive the solution for their parent. The detailed aggregation process is formalized in Algorithm 1. For parent

Algorithm 1 Aggregator Procedure

Input: Node n and LLM $\mathcal{L}$
Ensure: Synthesized solution for node n

```
PROCEDURE REDUCE(n, L)
if n is a leaf then
    return n.solved ? n.solution : NULL
end if
for all c ∈ n.children do
    if not c.solved then
        c.solution ← REDUCE(c, L)
        c.solved ← true
    end if
end for
S ← { c ∈ n.children | c.solved }
if S[0].created_by = PE then
    return S[0].solution
else
    R ← TopoSort({(c.solution, c.D_i) : c ∈ S})
    return L(φ_merge(n, R))
end if
```

nodes expanded via PE, the child's solution is directly adopted. For nodes expanded via PD, the Aggregator performs a more complex operation. It first constructs a directed acyclic graph (DAG) based on the declared dependencies and applies a topological sort. It then resolves the sub-problems in an order that respects these dependencies, feeding the solutions of pre-requisite problems into the context for solving dependent ones. Finally, it synthesizes the results to formulate a comprehensive solution for the parent node. This process continues until the root node is solved.

D. Model Analysis and Discussion

We analyze the computational cost of DVC and compare it with representative prompting baselines. Let n denote the maximum number of iterations, v the maximum number of verification attempts per transformation, L the number of leaf nodes in the final DAG, and I the number of internal PD nodes that require aggregation.

Computational Cost Model. The total number of LLM calls in DVC can be decomposed into three phases. (1) *Problem structuring*: one call for initial problem extraction. (2) *DAG construction*: at most n iterations, each comprising one policy-selection call, up to v transformation calls, and up to v verification calls, leading to $O(n(1 + 2v))$ calls. (3) *Solution phase*: L calls to solve leaf nodes and at most I calls for PD aggregation. Therefore, the overall cost is $O(1 + n(1 + 2v) + L + I)$. In typical settings (e.g., $n = 10, v = 3$), the construction phase dominates, resulting in approximately 70 LLM calls in the worst case.

Comparison with Existing Methods. Table I summarizes the computational characteristics of different prompting schemes. We define *verification coverage* as the fraction of reasoning steps that undergo explicit correctness checking.

Cost–Accuracy Trade-off. Although DVC introduces higher computational overhead than single-chain prompting, the additional cost is warranted for two reasons. First, verification reduces error propagation, as indicated by the 3.33% accuracy drop on AIME2025 when verification is removed in the ablation study. Second, the DAG structure enables efficient information reuse by aggregating solutions of subproblems in topological order, thereby mitigating redundant computation. In contrast to GoT's exponential worst-case complexity $O(b^d + m)$, DVC scales linearly with the iteration budget, yielding more predictable and controllable resource usage. Moreover, when all leaf nodes are solved early, the procedure can terminate before reaching the worst-case bound.

TABLE I
COMPUTATIONAL COST COMPARISON. k: SAMPLING COUNT, b: BRANCHING FACTOR, d: DEPTH, m: MERGE OPERATIONS, n: MAX ITERATIONS, v: VERIFICATION ATTEMPTS, L: LEAF NODES.

Method	LLM Calls	Verification	Structure
CoT	$O(1)$	None	Chain
CoT-SC	$O(k)$	Voting	k Chains
ToT	$O(b^d)$	Heuristic	Tree
GoT	$O(b^d + m)$	None	Graph
DVC	$O(nv + L)$	Full	DAG

IV. EXPERIMENTS

We design a comprehensive experimental evaluation to validate the effectiveness of DVC. First, we compare DVC against multiple baseline methods including CoT-SC, ToT, and GoT across three challenging benchmarks (AIME2024, AIME2025, and GPQA-Diamond) to demonstrate its superior reasoning performance. Second, we perform ablation studies by systematically removing the Verifier and Planner modules to quantify their individual contributions. Finally, we examine the effect of the max_iteration hyperparameter to understand the trade-off between decomposition depth and accuracy.

A. Experimental Setup

Datasets. We evaluate DVC on three challenging scientific reasoning benchmarks: AIME2024 and AIME2025 (American Invitational Mathematics Examination) requiring sophisticated mathematical reasoning and exact numerical answers, and GPQA-Diamond containing graduate-level multiple-choice questions in physics, chemistry, and biology [30].

Comparing Methods. (1) **Baseline** [31] employs minimal instructions. This prompting method demonstrates performance consistent with zero-shot Chain-of-Thought. (2) **CoT-SC** [16] generates multiple reasoning paths and selects the most consistent answer through majority voting to improve reliability. (3) **ToT** [17] explores a branching tree of intermediate reasoning steps, allowing lookahead and backtracking akin to search algorithms. (4) **GoT** [18] extends tree-based reasoning to a graph structure, enabling more flexible thought organization and aggregation of reasoning paths.

Model Setup. To ensure comprehensive and fair evaluation, we conduct experiments on both the closed-source model GPT-4o and the open-source model DeepSeek-V3 [32] to demonstrate the effectiveness of our proposed method.

TABLE II
PERFORMANCE ON AIME AND GPQA-DIAMOND.

Tasks	DeepSeek-V3					GPT-4o				
	BL	CoT-SC	ToT	GoT	DVC	BL	CoT-SC	ToT	GoT	DVC
AIME2024	30.00	36.67	26.67	40.00	**40.00**	13.33	13.33	13.33	13.33	**16.67**
AIME2025	20.00	30.00	26.67	33.33	**33.33**	3.33	6.67	3.33	10.00	**13.33**
GPQA-Diamond	58.58	61.11	58.08	63.63	**65.15**	50.00	52.02	50.50	54.54	**54.54**

TABLE III
PERFORMANCE COMPARISON ON GPQA DATASET ACROSS PHYSICS, CHEMISTRY, AND BIOLOGY.

Method	Phy.	Chem.	Bio.	All
Baseline				
DeepSeek-V3	73.25	45.16	57.89	58.58
GPT-4o	55.81	41.93	63.15	50.00
CoT-SC				
DeepSeek-V3	75.58	48.38	57.89	61.11
GPT-4o	63.95	40.86	52.63	52.02
ToT				
DeepSeek-V3	73.25	44.08	57.89	58.08
GPT-4o	56.97	41.93	63.15	50.50
GoT				
DeepSeek-V3	79.07	50.53	57.89	63.63
GPT-4o	66.28	41.93	63.15	54.54
DVC				
DeepSeek-V3	**79.07**	**53.76**	**57.89**	**65.15**
GPT-4o	**65.11**	**44.08**	**57.89**	**54.54**

B. Main Results

Table II and Table III summarize the performance of DVC and competing baselines across datasets and language models. Relative to CoT-SC, DVC improves accuracy by 3.33% on AIME2024, 3.33% on AIME2025, and 4.04% on GPQA-Diamond for DeepSeek-V3. For GPT-4o, DVC yields gains of 3.34% on AIME2024, 6.66% on AIME2025, and 2.52% on GPQA-Diamond compared with CoT-SC.

Compared with ToT, DVC achieves higher accuracy across all evaluated settings, with the largest improvement observed on GPQA-Diamond for DeepSeek-V3 (from 58.08% to 65.15%). DVC also improves upon GoT, another graph-based reasoning baseline. On GPQA-Diamond with DeepSeek-V3, DVC attains 65.15% versus 63.63% for GoT. On AIME2025 with GPT-4o, DVC improves from 10.00% (GoT) to 13.33%.

Table III further reports per-domain results on GPQA. DVC yields consistent gains across Physics, Chemistry, and Biology, with the largest improvement on Chemistry for DeepSeek-V3 (53.76% versus 50.53% for GoT). Overall, these results suggest that verification-guided DAG construction can improve robustness by reducing error propagation in multi-step reasoning.

C. Ablation Studies

We conduct ablation studies to quantify the contribution of key components in DVC.

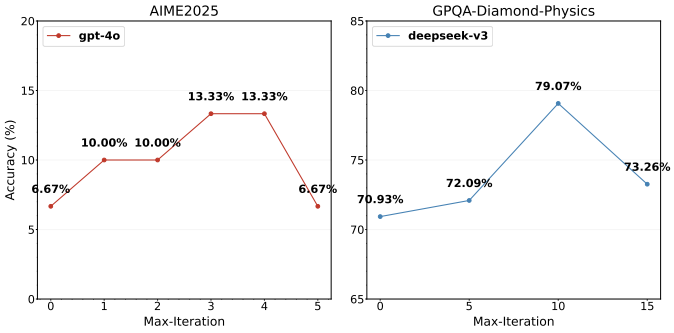

Fig. 2. Effect of `max_iteration` on accuracy for two benchmarks.

TABLE IV
ABLATION STUDY OF THE MAJOR COMPONENTS: VERIFIER AND PLANNER. ✗ MEANS WE DO NOT EMPLOY THE METHOD, AND ✓ MEANS WE USE THE METHOD.

Verifier	Planner	AIME2025	GPQA-Chemistry
✗	✗	6.67	36.55
✗	✓	10.00	39.78
✓	✓	**13.33**	**44.08**

Effect of the Verifier and Planner. Table IV reports results on AIME2025 and the Chemistry subset of GPQA-Diamond using GPT-4o. Removing both the Verifier and Planner degrades performance to 6.67% on AIME2025 and 36.55% on GPQA-Diamond (Chemistry). Adding the Planner alone improves accuracy to 10.00% and 39.78%, respectively, suggesting the benefit of informed transformation selection. The full DVC configuration achieves the best performance (13.33% on AIME2025 and 44.08% on GPQA-Diamond (Chemistry)), indicating that verification is critical for ensuring transformation correctness and mitigating error propagation.

Effect of Max Iteration. To assess the impact of iterative decomposition depth, we vary the `max_iteration` hyperparameter, which controls the maximum depth of the solution DAG. Results are shown in Fig. 2. For DeepSeek-V3 on the Physics subset of GPQA-Diamond, accuracy increases from 70.93% at `max_iteration`=0 to a peak of 79.07% at `max_iteration`=10. For GPT-4o on AIME2025, performance peaks at 13.33% for `max_iteration` values of 3 and 4, compared with 6.67% at `max_iteration`=0. Overall, these results highlight the utility of iterative decomposition while suggesting the existence of a task-dependent optimal depth that balances simplification against potential error intro-

duction.

V. CONCLUSION

This paper presents DVC, a training-free framework that improves the complex reasoning capability of large language models via a "Divide, Verify, and Conquer" strategy. DVC is built upon two fundamental operations PE and PD to structure complex tasks as a directed acyclic graph, and it integrates dedicated verification modules with multi-round reflection to promote logical soundness at each transformation step. Experimental results show that DVC yields consistent improvements on challenging mathematical and scientific reasoning benchmarks. In particular, DVC outperforms representative baselines, including CoT-SC, ToT, and GoT, on both AIME and GPQA-Diamond. Ablation studies further corroborate the contribution of the Verifier and Planner modules, indicating that verification-guided reasoning is critical for mitigating error propagation and improving reliability.

Future work includes incorporating memory mechanisms [33], [34] to inform transformations with accumulated experience, developing adaptive policies to select iteration depth dynamically, and integrating DVC with retrieval-augmented generation [35]–[37]to further improve performance across diverse scientific domains.

ACKNOWLEDGEMENTS

This work was supported by the National Natural Science Foundation of China (No. 62576299, No. U21B2037, No. U22B2051, No. U23A20383, No. 62176222, No. 62176223, No. 62176226, No. 62072386, No. 62072387, No. 62072389, No. 62002305, No. 62272401), the Natural Science Foundation of Fujian Province of China (No. 2021J06003, No. 2022J06001), and the Fundamental Research Funds for the Central Universities.

REFERENCES

[1] Z. Li, D. Zhang, M. Zhang, J. Zhang, Z. Liu, Y. Yao, H. Xu, J. Zheng, P. Wang, X. Chen, Y. Zhang, F. Yin, J. Dong, Z. Li, B. Bi, L. Mei, J. Fang, X. Liang, Z. Guo, L. Song, and C. Liu, "From system 1 to system 2: A survey of reasoning large language models," *arXiv preprint arXiv:2502.17419*, 2025.

[2] A. Forootani, "A survey on mathematical reasoning and optimization with large language models," *arXiv preprint arXiv:2503.17726*, 2025.

[3] J. Huang and K. Chang, "Towards reasoning in large language models: A survey," *arXiv preprint arXiv:2212.10403*, 2022.

[4] R. Liang, T. Zhang, Y. Lu, Y. Liu, Z. Huang, and X. Chen, "Astbert: Enabling language model for financial code understanding with abstract syntax trees," in *Proceedings of the Fourth Workshop on Financial Technology and Natural Language Processing (FinNLP)*, 2022, pp. 10–17.

[5] Y. Yan, J. Su, J. He, F. Fu, X. Zheng, Y. Lyu, K. Wang, S. Wang, Q. Wen, and X. Hu, "A survey of mathematical reasoning in the era of multimodal large language models: Benchmark, method & challenges," *arXiv preprint arXiv:2412.11936*, 2024.

[6] Z. Yu, R. Peng, K. Ding, Y. Li, Z. Peng, M. Liu, Y. Zhang, Z. Yuan, H. Xin, W. Huang *et al.*, "Formalmath: Benchmarking formal mathematical reasoning of large language models," *arXiv preprint arXiv:2505.02735*, 2025.

[7] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[8] C. Zheng, S. Liu, M. Li, X. Chen, B. Yu, C. Gao, K. Dang, Y. Liu, R. Men, A. Yang *et al.*, "Group sequence policy optimization," *arXiv preprint arXiv:2507.18071*, 2025.

[9] Y. Liu, T. Liu, T. Zhang, Y. Xia, J. Wang, Z. Shen, J. Jin, Z. Ding, and F. R. Yu, "Grl-prompt: Towards prompts optimization via graph-empowered reinforcement learning using llms' feedback," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2025, pp. 426–438.

[10] S. S. Srivastava and V. Aggarwal, "A technical survey of reinforcement learning techniques for large language models," *arXiv preprint arXiv:2507.04136*, 2025.

[11] K. Zhang, Y. Zuo, B. He, Y. Sun, R. Liu, C. Jiang, Y. Fan, K. Tian, G. Jia, P. Li *et al.*, "A survey of reinforcement learning for large reasoning models," *arXiv preprint arXiv:2509.08827*, 2025.

[12] Q. Dang and C. Ngo, "Reinforcement learning for reasoning in small llms: What works and what doesn't," *arXiv preprint arXiv:2503.16219*, 2025.

[13] Y. Liu, S. Chu, T. Zhang, H. Zhou, Z. Shen, J. Wang, J. Qi, and F. Xia, "Ml-ecs: A collaborative multimodal learning framework for edge-cloud synergies," *arXiv preprint arXiv:2602.14107*, 2026.

[14] Y. Liu, Y. Wang, T. Zhang, Z. Shen, C. Peng, L. Wu, F. Xia, and J. Jin, "A structure-agnostic co-tuning framework for llms and slms in cloud-edge systems," *arXiv preprint arXiv:2511.11678*, 2025.

[15] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

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

[17] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," *Advances in Neural Information Processing Systems*, vol. 36, pp. 11 809–11 822, 2023.

[18] M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gianinazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczyk, and T. Hoefler, "Graph of thoughts: Solving elaborate problems with large language models," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, 2024, pp. 17 682–17 690.

[19] S. Mukherjee, A. Chinta, T. Kim, T. Sharma, and D. Hakkani-Tür, "Premise-augmented reasoning chains improve error identification in math reasoning with llms," *arXiv preprint arXiv:2502.02362*, 2025.

[20] B. Wang, S. Min, X. Deng, J. Shen, Y. Wu, L. Zettlemoyer, and H. Sun, "Towards understanding chain-of-thought prompting: An empirical study of what matters," *arXiv preprint arXiv:2212.10001*, 2022.

[21] Z. Ling, Y. Fang, X. Li, Z. Huang, M. Lee, R. Memisevic, and H. Su, "Deductive verification of chain-of-thought reasoning," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 36 407–36 433.

[22] F. Barez, T. Wu, I. Arcuschin, M. Lan, V. Wang, N. Siegel, N. Collignon, C. Neo, I. Lee, A. Paren *et al.*, "Chain-of-thought is not explainability," *Preprint, alphaXiv*, p. v2, 2025.

[23] J. Li, Y. Fu, L. Fan, J. Liu, Y. Shu, C. Qin, M. Yang, I. King, and R. Ying, "Implicit reasoning in large language models: A comprehensive survey," *arXiv preprint arXiv:2509.02350*, 2025.

[24] X. Liu, "TOT-DS: demonstration selection for enhancing the reasoning capabilities of large language models," in *International Joint Conference on Neural Networks, IJCNN 2025, Rome, Italy, June 30 - July 5, 2025*. IEEE, 2025, pp. 1–8.

[25] Q. Zhu, "SRF - A structured method for enhancing large model reasoning," in *International Joint Conference on Neural Networks, IJCNN 2025, Rome, Italy, June 30 - July 5, 2025*. IEEE, 2025, pp. 1–7.

[26] Q. Huang, C. Jin, S. He, L. Shu, and Y. Long, "Kolmogorov-arnold network-enhanced cross-modal alignment for textual attribute graph reasoning with large language models," in *International Joint Conference on Neural Networks, IJCNN 2025, Rome, Italy, June 30 - July 5, 2025*. IEEE, 2025, pp. 1–8.

[27] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe, "Let's verify step by step," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=v8L0pN6EOi

[28] X. Zheng, J. Lou, B. Cao, X. Wen, Y. Ji, H. Lin, Y. Lu, X. Han, D. Zhang, and L. Sun, "Critic-cot: Boosting the reasoning abilities of large language model via chain-of-thoughts critic," *CoRR*, vol. abs/2408.16326, 2024.
[29] J. R. Chowdhury and C. Caragea, "Zero-shot verification-guided chain of thoughts," *ArXiv*, vol. abs/2501.13122, 2025.
[30] D. Rein, B. Li, A. Stickland, J. Petty, R. Pang, J. Dirani, J. Michael, and S. Bowman, "Gpqa: A graduate-level google-proof q&a benchmark," in *First Conference on Language Modeling*, 2024.
[31] M. Suzgun, M. Yuksekgonul, F. Bianchi, D. Jurafsky, and J. Zou, "Dynamic cheatsheet: Test-time learning with adaptive memory," *arXiv preprint arXiv:2504.07952*, 2025.
[32] DeepSeek-AI, "Deepseek-v3 technical report," *CoRR*, vol. abs/2412.19437, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2412.19437
[33] B. Wang, X. Wu, and Z. Li, "Informative memory mechanism for enhancing crisis response in large language models," in *International Joint Conference on Neural Networks, IJCNN 2025, Rome, Italy, June 30 - July 5, 2025*. IEEE, 2025, pp. 1–9. [Online]. Available: https://doi.org/10.1109/IJCNN64981.2025.11228206
[34] G. Hao, Y. Zhang, G. Ma, Y. Chen, F. Alexandre, and S. Yu, "Position paper: Large language models need episodic memory," in *International Joint Conference on Neural Networks, IJCNN 2025, Rome, Italy, June 30 - July 5, 2025*. IEEE, 2025, pp. 1–10. [Online]. Available: https://doi.org/10.1109/IJCNN64981.2025.11229266
[35] S. Li, Y. Yu, H. Yang, Y. Zhou, and C. Ji, "HIM-RAG: A heuristic framework for iterative multi-source retrieval-augmented generation," in *International Joint Conference on Neural Networks, IJCNN 2025, Rome, Italy, June 30 - July 5, 2025*. IEEE, 2025, pp. 1–8. [Online]. Available: https://doi.org/10.1109/IJCNN64981.2025.11229295
[36] H. Wang, Z. Qi, N. Wang, Z. Yu, Z. Li, and W. Lv, "SSR-RAG: A smart retrieval-augmented generation framework with rollback mechanisms for safer and more accurate answers," in *International Joint Conference on Neural Networks, IJCNN 2025, Rome, Italy, June 30 - July 5, 2025*. IEEE, 2025, pp. 1–7. [Online]. Available: https://doi.org/10.1109/IJCNN64981.2025.11227721
[37] F. Sanguigni, D. Morelli, M. Cornia, and R. Cucchiara, "Fashion-rag: Multimodal fashion image editing via retrieval-augmented generation," in *International Joint Conference on Neural Networks, IJCNN 2025, Rome, Italy, June 30 - July 5, 2025*. IEEE, 2025, pp. 1–8. [Online]. Available: https://doi.org/10.1109/IJCNN64981.2025.11229186

APPENDIX

This appendix provides the detailed prompt templates used in the DVC framework for all major modules: Problem Structurer, Planner, PE, PD, Verifier, Solver, and Aggregator.

A. Prompt for Problem Structurer

Please convert the following problem into a structured format with clear conditions and objectives.

Original problem text:
{problem_text}

Guidelines:
1. Extract all given conditions/constraints from the problem
2. Identify the specific objectives/goals to be achieved
3. Each condition should be a clear, standalone statement
4. Each objective should be a specific task or question to answer
5. Preserve all important information from the original problem
6. Use clear, mathematical/scientific language appropriate for the domain

B. Prompt for Planner

Given this problem:
{node_state}

I need to decide whether to:
1. Transform this problem into an equivalent but simpler form (Problem Equivalence)
2. Decompose this problem into smaller sub-problems (Problem Decomposition)

Consider:
- How complex is the current formulation?
- Are there obvious simplifications (algebraic, logical, conceptual)?
- Can the problem be naturally split into cases or sub-problems?
- Which approach is more likely to make progress toward a solution?

Respond with ONLY "PE" or "PD".

C. Prompt for PE

Transform this problem into an equivalent but simpler form.

Feedback from the Verifier:
{feedback_section}

ORIGINAL PROBLEM:
{node_state}

Rules:
1. The new formulation must be equivalent to the original within the problem's domain.
2. The transformation should make the problem easier to solve.
3. Use algebraic manipulation, logical equivalence, or other relevant scientific principles.
4. Clearly explain your transformation steps.

ABSOLUTELY FORBIDDEN:
- DO NOT reference parent conditions or objectives by number (e.g., "Condition 1", "Condition 2", "Objective 1")
- DO NOT write things like "Condition 1: Condition 2" or "Objective 1: Objective 1"
- EVERY condition and objective must contain the FULL, COMPLETE content, not references
- Write out the complete text of each condition and objective in your transformation

D. Prompt for PD

Decompose this problem into smaller, independent, and self-contained sub-problems.

Feedback from the Verifier:
{feedback_section}

ORIGINAL PROBLEM:
{node_state}

CRITICAL REQUIREMENTS:
1. Each sub-problem MUST be COMPLETELY SELF-CONTAINED with ALL information needed to solve it.
2. Each sub-problem must include ALL relevant conditions from the original problem needed for its solution.
3. Sub-problems must be INDEPENDENTLY SOLVABLE. If a

sub-problem needs results from other sub-problems, explicitly list their IDs in the field ‘depends_on‘.
4. Do NOT create hidden dependencies – any dependency MUST be declared via ‘depends_on‘.
5. If information must be shared, DUPLICATE it in each relevant sub-problem.
6. The collection of sub-problems must completely cover the original problem’s scope.

ABSOLUTELY FORBIDDEN:
- DO NOT reference parent conditions or objectives by number (e.g., ”Condition 1”, ”Condition 2”, ”Objective 1”)
- DO NOT write things like ”Condition 1: Condition 2” or ”Objective 1: Objective 1”
- EVERY condition and objective must contain the FULL, COMPLETE content, not references
- Write out the complete text of each condition and objective in your sub-problems

DECOMPOSITION STRATEGIES:
- Case analysis: Break down by different cases or scenarios
- Divide and conquer: Split into independent components
- Step-by-step: Create sequential steps only if each step is self-contained
- Analysis by parts: Decompose by different parts of a complex structure

E. Prompt for Verifier

Verify Equivalence (for PE):

Verify if this transformation preserves equivalence for this problem:

PARENT PROBLEM:
{parent_node_state}

TRANSFORMED PROBLEM (CHILD NODE):
{transformed_node_state}

Perform a step-by-step verification:
1. Check if every solution to the parent problem is also a solution to the transformed child problem.
2. Check if every solution to the transformed child problem is also a solution to the parent problem.
3. Verify that no constraints were incorrectly added or removed.
4. Confirm that the transformation is logically and scientifically sound for the given domain.

Verify Decomposition (for PD):

Verify if this problem decomposition is complete and correct for this problem:

PARENT PROBLEM:
{parent_node_state}

DECOMPOSED INTO:
{sub_problems_desc}

Perform a step-by-step verification:
1. Check if the sub-problems together cover all aspects of the parent problem.
2. Check if solving all sub-problems would lead to a solution of the parent problem.
3. Verify that no important constraints or conditions were lost.
4. Confirm that the decomposition is logically and scientifically sound for the given domain.

F. Prompt for Solver

You MUST solve this problem directly.
Even if it seems difficult, make your best attempt.

Problem:
{node_state}

Rules:
1. Provide a complete, detailed solution with clear steps.
2. Break down complex parts if necessary.
3. Give a final, precise answer.
4. Format your answer clearly.

G. Prompt for Aggregator

I have solutions to parts of a problem:

Original problem:
{node_state}

Sub-problems and their solutions:
{child_results_str}

Please combine these solutions to answer the original problem. Consider:
1. If the sub-problems are from Problem Decomposition (PD), you need to combine their results.
2. If the sub-problems are from Problem Equivalence (PE), the child solution should already solve the original problem, but verify this.

Give a clear, final answer to the original problem.