Dual-Cluster Memory Agent: Resolving Multi-Paradigm Ambiguity in Optimization Problem Solving

Xinyu Zhang[1,2*], **Yuchen Wan**[1,2*], **Boxuan Zhang**[1,2], **Zesheng Yang**[1,2], **Lingling Zhang**[1,2†], **Bifan Wei**[1,2], **Jun Liu**[1,3]
[1]School of Computer Science and Technology, Xi'an Jiaotong University
[2]Ministry of Education Key Laboratory of Intelligent Networks and Network Security, China
[3]Shaanxi Province Key Laboratory of Big Data Knowledge Engineering, China
zhang1393869716@stu.xjtu.edu.cn, {zhanglling,liukeen}@xjtu.edu.cn

Abstract

Large Language Models (LLMs) often struggle with structural ambiguity in optimization problems, where a single problem admits multiple related but conflicting modeling paradigms, hindering effective solution generation. To address this, we propose **Dual-Cluster Memory Agent (DCM-Agent)** to enhance performance by leveraging historical solutions in a training-free manner. Central to this is Dual-Cluster Memory Construction. This agent assigns historical solutions to modeling and coding clusters, then distills each cluster's content into three structured types: *Approach*, *Checklist*, and *Pitfall*. This process derives generalizable guidance knowledge. Furthermore, this agent introduces **Memory-augmented Inference** to dynamically navigate solution paths, detect and repair errors, and adaptively switch reasoning paths with structured knowledge. The experiments across seven optimization benchmarks demonstrate that DCM-Agent achieves an average performance improvement of 11%- 21%. Notably, our analysis reveals a "knowledge inheritance" phenomenon: memory constructed by larger models can guide smaller models toward superior performance, highlighting the framework's scalability and efficiency.

1 Introduction

Optimization problems underpin operations research, supporting applications from supply chain logistics to economic forecasting (Liu et al., 2025b, 2023; Belil et al., 2018). Traditionally, solving these problems requires an intensive process (Meerschaert, 2013), where domain experts manually translate textual descriptions into formulations. Recently, Large Language Models (LLMs) have demonstrated remarkable reasoning capabilities across diverse domains, including science reasoning (Zhang et al., 2025a,d,e), cognitive reasoning

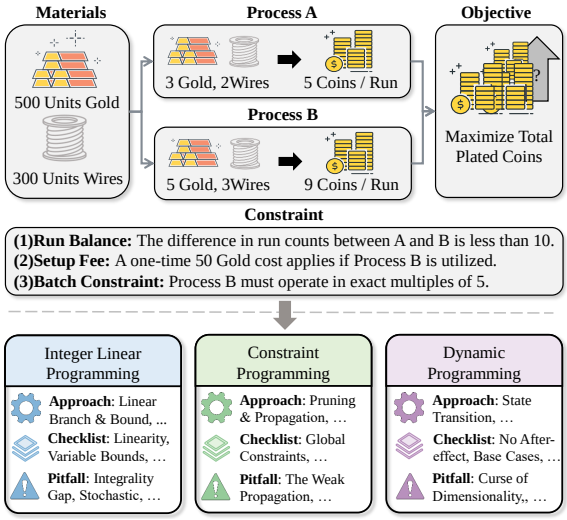

Figure 1: Illustration of a single production planning problem formalized via distinct algorithmic paradigms.

(Zhang et al., 2025c), and temporal analysis (Zhang et al., 2022). These advances have also reshaped optimization modeling, leveraging domain knowledge to automate the conversion of textual descriptions into optimization models, thereby mitigating dependency on human expertise (Sinha et al., 2025; Zhao and Cheong, 2025). However, harnessing LLMs for this purpose remains a challenge, leading to diverse preliminary explorations (Wang et al., 2025; Jiang et al., 2025a; Chen et al., 2025b).

A fundamental bottleneck impeding current methods is cognitive interference from entangled modeling paradigms. As shown in Figure 1, this presents conflicting signals, where logical constraints (e.g., exact multiples) suggest Constraint Programming (CP), resource maximization implies Integer Linear Programming (ILP), and sequential stage dependencies signal Dynamic Programming (DP). While these features are relevant, their simultaneous presence acts as a confounding distractor.

This complexity exposes deficiencies in current paradigms. Fine-tuning approaches are misled by this interference, mechanically applying memorized templates when subtle variations necessitate

*These authors contributed equally to this work.
†Corresponding author

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 5895–5908
July 2-7, 2026

alternative strategies (Wang and Li, 2025). Agentic frameworks struggle with these trade-offs during verification; facing ambiguous algorithmic choices, they rely on the static prompts (e.g., "Check if correct") that lack the granularity to detect paradigm-specific pitfalls (Huang et al., 2024), such as distinguishing between linearity gaps in ILP and recurrence validity in DP (Zhang and Luo, 2025). This reveals a fundamental tension: the interference problem demands flexibility to navigate paradigm ambiguity, yet also requires targeted knowledge to verify paradigm-specific correctness.

To address this, we propose the **Dual-Cluster Memory Agent (DCM-Agent)**, a training-free framework that balances flexibility and structure by externalizing reasoning patterns into historical archives. Instead of relying on parameter updates, this framework decouples abstract modeling from precise coding by directly capitalizing on historical solution archives. Central to this framework is the **Dual-Cluster Construction**, which resolves structural ambiguity by organizing historical data, ranging from canonical successes to persistent failures, into a bipartite graph that quantifies the relationships between the independent *Modeling Cluster* and the *Coding Cluster*. This process distills the raw experience nodes within each cluster into three cluster-level structured knowledge tiers: *Approach*, *Checklist*, and *Pitfall*. By explicitly mapping the decision space, this design enables the agent to filter out interfering noise and translate isolated solutions into robust structural knowledge.

Building upon the dual-cluster memory, we introduce **Memory-Augmented Inference**. Unlike conventional static prompting, this mechanism effectively solves new problems by retrieving relevant clusters that offer corresponding structured knowledge to guide. The inference is executed through an iterative generate-verify-repair-backtrack pipeline that operates dynamically. By leveraging the retrieved *Pitfalls* and *Checklists*, this pipeline systematically steers the generation process, allowing the agent to detect method-specific errors (e.g., integrality gaps versus recurrence failures) and autonomously backtrack to alternative reasoning paths when a chosen paradigm proves infeasible.

We evaluate DCM-Agent across seven diverse optimization benchmarks, where it achieves an average performance improvement of 11%–21% compared to standalone LLMs. The results demonstrate that DCM-Agent maintains consistent state-of-the-art accuracy across various model scales without the computational overhead of training, offering a superior trade-off between solution precision and efficiency. Crucially, our analysis reveals a "knowledge inheritance" phenomenon: memory constructed by larger models can be effectively transferred to guide smaller models toward superior performance, verifying the framework's scalability and the transferability of its training-free logic.

2 Related Work

2.1 LLM-based Optimization Modeling

LLM-based optimization modeling reduces the expertise required for complex formulation via prompt-based strategies or fine-tuning methods. Prompt-based frameworks utilize multi-agent workflows (Xiao et al., 2023; AhmadiTeshnizi et al., 2024; Zhang et al., 2025b) or tree-search algorithms (Liu et al., 2025c; Astorga et al., 2025) to improve reasoning and code generation in general-purpose LLMs. Conversely, fine-tuning methods like FOARL (Jiang et al., 2025b), ORLM (Huang et al., 2025a), and SIRL (Chen et al., 2025a) develop specialized models by training on domain-specific operations research datasets to internalize modeling patterns.

2.2 Retrieval-Augmented Reasoning

Despite the progress of reasoning LLMs (OpenAI, 2024; Guo et al., 2025), reliance on internal knowledge often leads to hallucinations that prompting alone cannot resolve (Huang et al., 2025b; Wei et al., 2022). Retrieval-Augmented Reasoning addresses this by integrating external verification into the reasoning process (Barry et al., 2025). While frameworks like ReAct (Yao et al., 2022) and Retrieval-Augmented Thoughts (Wang et al., 2024) dynamically revise traces with retrieved data, this paradigm is also effective for optimization. For example, OptiTree (Liu et al., 2025c) retrieves analogous subproblems to ground reasoning, ensuring complex modeling steps remain verifiable.

3 Methodology

3.1 Overview

We formalize optimization problem solving via LLMs as a composite structured process spanning the problem space $\mathcal{X}$, modeling space $\mathbb{M}$, and coding space $\mathbb{C}$. The solution $\hat{y}$ is derived via:

$$\hat{y} = \mathcal{E}(\underbrace{h_\psi(c \mid m)}_{\text{Coding}} \circ \underbrace{g_\phi(m \mid x)}_{\text{Modeling}}) \quad (1)$$

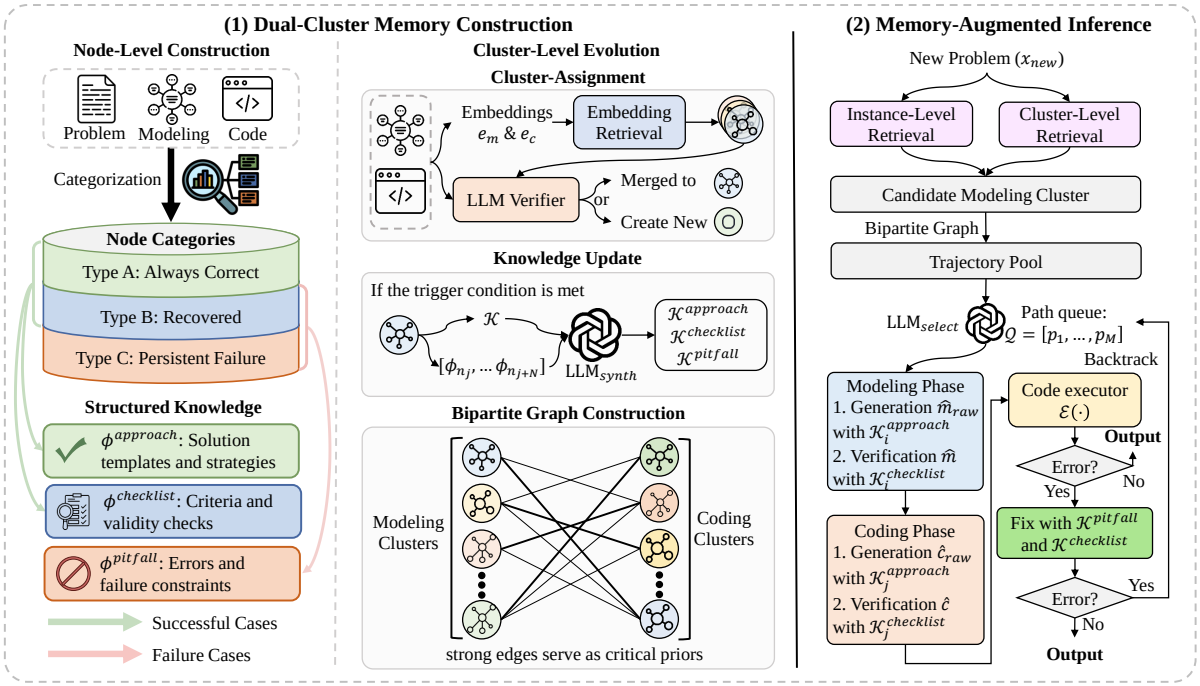

Figure 2: Overview of the Dual-Cluster Memory Agent (DCM-Agent). This agent operates in two distinct phases: (1) **Dual-Cluster Memory Construction:** Historical solutions are stratified into three types to distill structured knowledge $\mathcal{K}$. These are organized into decoupled Modeling Clusters and Coding Clusters, bridged by a weighted bipartite graph $\mathcal{G}$. (2) **Memory-Augmented Inference:** For a new problem, the agent retrieves relevant cluster paths to guide the sequential generation of the mathematical formulation $\hat{m}$ and executable code $\hat{c}$.

where g_ϕ generates the modeling logic $\hat{m}$, h_ψ synthesizes the executable code $\hat{c}$, and $\mathcal{E}(\cdot)$ serves as the code executor. The core challenge lies in the intrinsic one-to-many nature of this $x \to \hat{m} \to \hat{c}$ mapping: as a single problem admits diverse valid modeling logics and coding implementation (Figure 1), the model must possess robust judgment capabilities to navigate these possibilities.

We propose the **Dual-Cluster Memory Agent (DCM-Agent)**, designed to manage this complexity by explicitly decoupling modeling (g_ϕ) from coding (h_ψ), as shown in Figure 2. DCM-Agent operates in two phases: (1) *Dual-Cluster Memory Construction*, which distills instance-level experience nodes into generalized structured knowledge, and (2) *Memory-Augmented Inference*, which applies the knowledge to solve novel problems.

To support this, DCM-Agent maintains a *dynamic memory* $\mathcal{D}$ with a novel hierarchical structure: specific experience nodes are organized into decoupled Modeling Clusters and Coding Clusters. While individual nodes store trajectory details, each cluster maintains a high-level generalized knowledge ($\mathcal{K}$) synthesized from its constituent nodes. At the same time, DCM-Agent constructs a specialized bipartite graph $\mathcal{G}$ to bridge these clusters, modeling the compatibility between abstract modeling logics and concrete coding strategies.

3.2 Dual-Cluster Memory Construction

This phase transforms raw problem-solution trajectories into a structured, decoupled memory system. The process follows a rigorous bottom-up lifecycle, progressing from the discrete analysis of individual instances to collective knowledge synthesis.

3.2.1 Node-Level Construction

We first curate a dataset of distinct samples, disjoint from the evaluation benchmarks. To ensure the quality of extracted knowledge, we classify the solutions of these samples into three categories:

- **Type A (Always Correct):** Samples are consistently solved correctly across multiple attempts. These represent canonical problem-solving patterns and serve as standard references.
- **Type B (Recovered):** The samples that initially failed but yielded correct solutions upon reattempting. These instances capture the boundary between failure and success, providing critical information on specific failure modes and their corresponding recovery reasoning.
- **Type C (Persistent Failure):** The solutions persist in failure despite exhaustive attempts (e.g.,

Guidance Tier	Source Type	Content Definition
ϕ^{approach} (How to solve)	Type A + Type B (Success)	Solution templates and logical steps
$\phi^{\text{checklist}}$ (What to verify)	Type A + Type B (Success)	Validity criteria and boundary checks
ϕ^{pitfall} (What to avoid)	Type B + Type C (Failure)	Common errors and constraint violations

Table 1: Mapping from sample types to guidance tiers.

exceeding 3 rounds). These encode fundamental mismatches between problems and approaches.

This stratification facilitates differential knowledge extraction: the successful instances (Types A and B) provide positive references and caveats, while failures (Types B and C) reveal critical pitfalls.

To transform these raw samples into actionable memory, we first decompose each problem-solution pair into distinct modeling logic and coding implementation components. For each component, we generate embeddings $(\mathbf{e}_m, \mathbf{e}_c)$ to enable semantic retrieval, and simultaneously extract a tuple of instance-specific knowledge, denoted as $\Phi_n = \langle \phi_n^{\text{approach}}, \phi_n^{\text{checklist}}, \phi_n^{\text{pitfall}} \rangle$, as summarized in Table 1. Here, Φ_n represents local insights specific to node n. Specifically, we synthesize canonical approaches (ϕ_n^{approach}) and verification checklists ($\phi_n^{\text{checklist}}$) from successful nodes (Types A and B), while deriving explicit pitfall warnings (ϕ_n^{pitfall}) from failure trajectories (Types B and C).

3.2.2 Cluster-Level Evolution

Once individual nodes are processed, DCM-Agent organizes them to form generalized knowledge. This involves clustering nodes, synthesizing knowledge, and establishing graph connectivity.

Cluster Assignment. Each experience node is integrated into the memory via a rigorous assignment process. First, embedding-based retrieval identifies top-k candidate clusters by comparing the node's embedding ($\mathbf{e}_m$ or $\mathbf{e}_c$) with cluster centroids $\boldsymbol{\mu}$. Second, an LLM-based verifier checks semantic consistency to either merge the node into a matched candidate or initialize a new cluster.

Knowledge Update. To resolve the ambiguity between the specific examples and general patterns, we employ an incremental update mechanism. Each cluster maintains a generalized knowledge $\mathcal{K} = \langle \mathcal{K}^{\text{approach}}, \mathcal{K}^{\text{checklist}}, \mathcal{K}^{\text{pitfall}} \rangle$, which serves as the consolidated schema for that cluster (distinct from the raw Φ_n of individual nodes). When a cluster accumulates a threshold of new nodes (such as

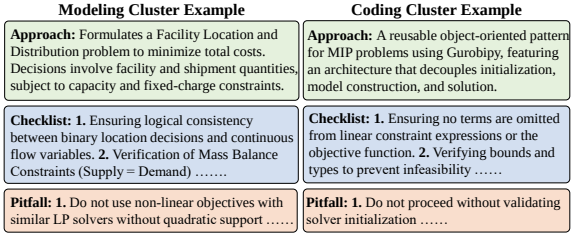

Figure 3: Two examples in our Dual-Cluster Memory.

$N = 5$), we trigger a knowledge update step:

$$\mathcal{K}^{(t+1)} = \text{LLM}_{\text{synth}} \left(\mathcal{K}^{(t)} \cup \bigcup_{j=1}^{N} \Phi_{n_j} \right) \quad (2)$$

Here, $\text{LLM}_{\text{synth}}$ is used to abstract generalized patterns from the new batch of instance knowledge Φ_{n_j} and merge them into the generalized knowledge $\mathcal{K}^{(t)}$. This ensures that $\mathcal{K}$ evolves to capture robust, non-redundant insights while retaining specific pitfall warnings, and is not overly influenced by extreme samples, as shown in Figure 3.

Bipartite Graph Construction. At the same time, we introduce a bipartite graph $\mathcal{G}$ to model the associations between the these decoupled clusters. Since each experience node n naturally maps to a pair of clusters (C_i^M, C_j^C), these linkages aggregate into a global structure. We formalize this as a bipartite graph $\mathcal{G} = (V_M, V_C, E)$, where the edge weight w_{ij} quantifies the co-occurrence frequency of modeling logic C_i^M and coding strategy C_j^C. The strong edges represent proven pathways, providing critical priors for subsequent usage.

3.3 Memory-Augmented Inference

3.3.1 Dual-Retrieval

For a new problem x_{new}, DCM-Agent leverages the memory to efficiently navigate the solution space by retrieving relevant historical experiences. We first encode the problem into the modeling logic embedding $\mathbf{e}_{\text{new}}$ and employ two complementary retrieval mechanisms to balance the problem relevance with general algorithmic applicability:

Instance-Level Retrieval captures the granular problem similarity by retrieving specific nodes $\mathcal{H}$ closest to $\mathbf{e}_{\text{new}}$, thereby identifying relevant experience nodes that share detailed semantic features:

$$\mathcal{H} = \arg\max_{K} \{ \text{sim}(\mathbf{e}_{\text{new}}, \mathbf{e}_i) \mid x_i \in \mathcal{D} \}. \quad (3)$$

Cluster-Level Retrieval targets abstract patterns by comparing $\mathbf{e}_{\text{new}}$ directly with cluster centroids

$\boldsymbol{\mu}_k^M$, which ensures capturing the modeling logic beyond surface-level textual matches:

$$\mathcal{S}_{\text{cluster}} = \arg\max_{K} \{\text{sim}(\mathbf{e}_{\text{new}}, \boldsymbol{\mu}_k^M)\} \quad (4)$$

These two sources are united to yield a robust final set of candidate modeling clusters, integrating both specific exemplars and general categories:

$$\mathcal{R} = \{C^M(x_i) \mid x_i \in \mathcal{H}\} \cup \mathcal{S}_{\text{cluster}}. \quad (5)$$

To effectively bridge modeling logic with coding implementation, we query the graph $\mathcal{G}$ to exploit learned associations. For each identified $C_i^M \in \mathcal{R}$, we retrieve the top-K coding neighbors $\mathcal{N}_k$ with the highest edge weights to form a diverse trajectory pool $\mathcal{P}$. Finally, an LLM selector serves as a verifier to rank these combinations based on their logical alignment with x_{new}, returning a prioritized queue $\mathcal{Q}$ of the most promising M solution paths:

$$\mathcal{Q} = \underset{(p)\in\mathcal{P}}{\text{Top-}M} \left(\text{LLM}_{\text{select}}(\mathcal{P}, x_{\text{new}})\right) \quad (6)$$

3.3.2 Solving via Generalized Knowledge

DCM-Agent processes the prioritized queue $\mathcal{Q}$ using a Generate-Verify-Repair-Backtrack pipeline. Crucially, all the steps are conditioned on the $\mathcal{K}$ of the selected clusters, rather than node knowledge ϕ, as shown in Figure 2. For a path $p_t = (C_i^M, C_j^C)$:

1. Generation & Verification. The LLM generates the modeling logic $\hat{m}_{\text{raw}}$ using the cluster's canonical approach $\mathcal{K}_i^{\text{approach}}$. Immediately, it verifies $\hat{m}_{\text{raw}}$ against the cluster's checklist $\mathcal{K}_i^{\text{checklist}}$:

$$\hat{m}_{\text{raw}} = \text{LLM}_{\text{gen}}(x_{\text{new}} \mid \mathcal{K}_i^{\text{approach}}) \quad (7)$$

$$\hat{m} = \text{LLM}_{\text{verify}}(\hat{m}_{\text{raw}} \mid \mathcal{K}_i^{\text{checklist}}) \quad (8)$$

Once the $\hat{m}$ is established, DCM-Agent transitions to code generation. The executable code $\hat{c}$ is generated using the coding cluster's templates $\mathcal{K}_j^{\text{approach}}$ and rigorously checked against guidelines $\mathcal{K}_j^{\text{checklist}}$, ensuring the robustness of the code structure

2. Repair & Backtracking. If the execution $\mathcal{E}(\hat{c})$ fails, resulting in the runtime error e, DCM-Agent initiates a knowledge-guided repair mechanism. Instead of blind debugging, it systematically analyzes e in the context of the cluster's specific pitfall warnings $\mathcal{K}_j^{\text{pitfall}}$, which contain common error patterns associated with this algorithm type:

$$\hat{c}_{\text{fixed}} = \text{LLM}_{\text{fix}}(\hat{c}, e \mid \mathcal{K}_j^{\text{pitfall}}, \mathcal{K}_j^{\text{checklist}}) \quad (9)$$

If repair attempts fail to yield a solution within a limit, the agent triggers a backtracking protocol. It discards the path and activates the next p_{t+1} from $\mathcal{Q}$, preventing the system from getting stuck in local optima and ensuring robust problem-solving.

4 Experiment

4.1 Experiment Setups

Datasets. To comprehensively evaluate our DCM-Agent framework across a spectrum of complexities, we utilize a diverse suite of seven optimization benchmarks. We employ NL4Opt (Ramamonjison et al., 2023) and NLP4LP (AhmadiTeshnizi et al., 2024) as standard baselines for linear and mixed-integer programming. To ensure rigorous testing on more demanding tasks, we incorporate OptiBench (Yang et al., 2025b), OptMATH (Lu et al., 2025), and the ComplexLP subset of MAMO (Huang et al., 2025c). Finally, we assess real-world applicability using IndustryOR (Huang et al., 2025a) and ComplexOR (Xiao et al., 2023) datasets. The memory is constructed using 500 problems that do not intersect with the current benchmarks.

Baselines. We systematically compare DCM-Agent with diverse methods on different LLMs of varying sizes, ranging from generic standard LLMs to advanced specialized optimization frameworks. We first establish a fundamental baseline using LLMs of varying sizes, including Qwen3 series (8B, 30B, 235B) (Yang et al., 2025a), DeepSeek-V3.2 (Liu et al., 2025a), and GPT-5.1 (OpenAI, 2025). Subsequently, we compare against representative specialized methods: (1) OptiMUS (AhmadiTeshnizi et al., 2024), a multi-agent workflow that enhances reliability through structured input processing; (2) AF-MCTS (Astorga et al., 2025), which employs Monte Carlo Tree Search to sequentially identify variables, constraints, and objectives; and (3) OptiTree (Liu et al., 2025c), which tackles high-complexity tasks by adaptively decomposing problems into manageable sub-problems.

Evaluation. Consistent with prior research, we adopt strict *end-to-end solving accuracy* as our primary evaluation metric. Our protocol evaluates the complete resolution pipeline: given a problem, the model generates the code c to produce execution output $o = \text{Python}(c)$, where the allowed libraries are *Gurobi*, *PuLP*, *OR-Tools*, *SciPy*, and *NetworkX*. A problem is considered solved if the extracted numerical answers for both the requirement and the objective function match the ground truth.

Method	Datasets							Avg.
	NL4Opt	ComplexLP	NLP4LP	OptiBench	OptMATH	IndOR	ComplexOR	
Size	230	211	242	605	166	100	18	-
Qwen3-8B								
Baseline	41.74	16.11	35.12	30.58	10.24	19.00	22.22	27.99
OptiMUS	53.48	23.22	43.39	44.13	15.06	23.00	33.33	38.04
AF-MCTS	47.39	19.43	39.26	36.53	12.65	21.00	33.33	32.70
OptiTree	55.22	25.59	46.28	45.12	16.26	25.00	38.89	39.76
DCM-Agent	**64.35**	**32.23**	**62.40**	**55.37**	**21.69**	**30.00**	**50.00**	**49.43**
Qwen3-30B								
Baseline	55.22	28.44	52.07	43.64	16.87	25.00	38.89	40.52
OptiMUS	63.48	33.64	63.64	56.20	24.70	29.00	50.00	50.25
AF-MCTS	65.65	34.60	68.18	57.68	25.90	31.00	50.00	52.22
OptiTree	70.88	36.49	70.25	59.50	27.71	30.00	55.56	54.45
DCM-Agent	**77.39**	**41.23**	**76.03**	**64.30**	**32.53**	**34.00**	**61.11**	**59.61**
Qwen3-235B								
Baseline	78.13	40.76	74.38	60.83	31.33	32.00	55.56	57.74
OptiMUS	83.48	44.55	77.27	64.30	37.95	34.00	66.67	61.77
AF-MCTS	87.39	46.92	78.93	68.26	40.36	37.00	61.11	64.82
OptiTree	89.56	48.82	80.16	70.74	41.57	36.00	66.67	66.66
DCM-Agent	**93.48**	**54.76**	**84.71**	**75.21**	**46.39**	**40.00**	**72.22**	**71.28**
Deepseek-V3.2								
Baseline	82.61	38.39	71.73	58.68	36.75	34.00	61.11	57.61
OptiMUS	86.09	43.60	75.21	62.15	39.76	36.00	66.67	61.20
AF-MCTS	89.13	46.92	78.10	67.27	43.98	39.00	66.67	65.14
OptiTree	90.87	47.87	79.34	70.08	47.59	38.00	72.22	67.18
DCM-Agent	**95.22**	**53.55**	**83.06**	**74.05**	**52.73**	**41.00**	**77.77**	**71.47**
GPT5.1								
Baseline	87.39	55.45	84.30	68.26	43.38	44.00	66.67	67.62
OptiMUS	90.43	58.77	86.78	71.07	45.78	46.00	72.22	70.42
AF-MCTS	94.47	60.66	88.84	74.71	51.20	51.00	77.77	73.93
OptiTree	95.65	62.56	90.08	76.86	53.61	49.00	77.77	75.51
DCM-Agent	**97.83**	**65.40**	**93.39**	**80.16**	**55.42**	**53.00**	**83.33**	**78.50**

Table 2: Solving accuracy (%) comparison across different datasets. The best results are highlighted in **bold**.

Method	NLP4LP	OptiBench	OptMATH
Baseline	8.3s	13.7s	21.7s
OptiMUS	26.5s	46.6s	86.3s
AF-MCTS	85.3s	110.8s	205.7s
OptiTree	17.7s	33.5s	61.6s
DCM-Agent	22.1s	41.3s	73.4s

Table 3: Time cost statistics (in seconds) of the Qwen3-235B model across selected benchmarks.

Ratio	0%	10%	40%	70%	100%
NLP4LP	74.38	78.51	81.40	82.64	84.71
OptiBench	60.83	66.45	71.57	74.05	75.21
OptMATH	31.33	36.75	40.36	44.58	46.39

Table 4: Performance comparison under different memory budgets (10%, 40%, 70%, and 100%) with Qwen3-235B across selected datasets. The results demonstrate the impact of memory constraints on solving accuracy.

4.2 Main Results

By carefully analyzing the accuracy results in Table 2 alongside the detailed time cost statistics in Table 3, we derive the following key insights:

Superiority of Our DCM-Agent across Model Scales. DCM-Agent consistently achieves the highest accuracy across all evaluated model sizes, ranging from the 8B parameter scale to GPT5.1. This uniform success demonstrates the framework's robustness and its capability to serve as a universal performance enhancer for optimization tasks regardless of the underlying backbone architecture.

Lower Capability Requirements for Memory Construction. The performance gains of DCM-Agent are most pronounced on smaller LLMs because its structured memory construction process is relatively less demanding on the model's inherent

Memory	NLP4LP	OptiBench	OptMATH
Qwen3-8B			
Baseline	35.12	30.58	10.24
Qwen3-8B	62.40	55.37	21.69
Qwen3-30B	67.36	59.34	23.49
Qwen3-235B	**69.42**	**60.17**	**25.30**
DeepSeek-V3.2	66.53	58.68	24.10
GPT-5.1	65.28	57.52	22.89
Qwen3-235B			
Baseline	74.38	60.83	31.33
Qwen3-8B	78.51	69.59	40.96
Qwen3-30B	82.64	73.22	43.98
Qwen3-235B	84.71	75.21	46.39
DeepSeek-V3.2	85.54	76.03	47.59
GPT-5.1	**86.36**	**77.36**	**48.19**

Table 5: Cross-model knowledge transfer: Impact of memory construction model on performance (%).

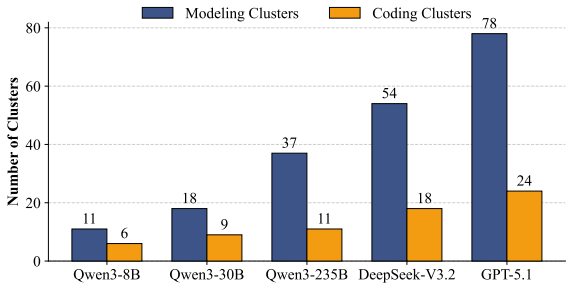

Figure 4: Statistical distribution of the number of two clusters obtained from LLMs of different sizes.

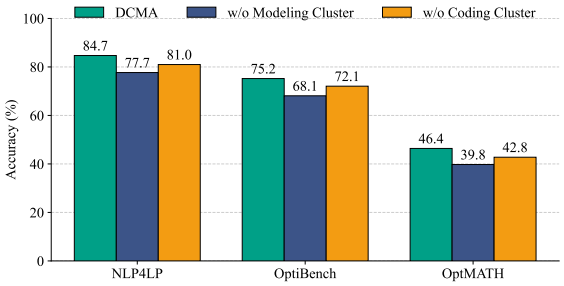

Figure 5: Ablation study on NLP4LP, OptiBench, and OptMATH datasets using Qwen3-235B.

Setting	NLP4LP	OptiBench	OptMATH
Retrieval Top-K in construction and inference			
$K = 1$	79.34	70.91	40.36
$K = 3$	84.71	75.21	46.39
$K = 5$	83.06	74.05	43.98
Memory Update Threshold (N) in construction			
$N = 1$	84.30	74.21	44.58
$N = 5$	84.71	75.21	46.39
$N = 10$	82.64	72.73	42.77
Planning Candidates (M) in inference			
$M = 1$	82.23	72.56	42.17
$M = 3$	84.71	75.21	46.39
$M = 5$	85.54	75.87	46.99

Table 6: Parameter sensitivity analysis on key hyperparameters (K, M, N) across three benchmarks.

reasoning power. By offloading constraint management to an external memory module, DCM-Agent allows smaller models to overcome their parameter limitations and thereby achieve competitive results typically reserved for much larger models.

Balanced Efficiency and Computational Overhead. As shown in Table 3, DCM-Agent significantly reduces the cost time compared to heavy search-based methods like AF-MCTS while maintaining superior accuracy. This suggests that DCM-Agent's memory-driven reasoning is more computationally purposeful than exhaustive tree exploration, providing an optimal overall trade-off between solving performance and time efficiency.

4.3 Dual-Cluster Memory Analysis

Cluster Statistics across LLMs. We investigate the number of Modeling Clusters (MC) and Constraint Clusters (CC) generated by different LLMs during the memory construction phase. As illustrated in Figure 4, stronger models (e.g., GPT-5.1) tend to generate a significantly higher number of Modeling Clusters compared to smaller models (e.g., Qwen3-8B). This suggests that advanced LLMs possess a more granular understanding of abstract problem structures, allowing them to disentangle subtle differences in mathematical formulations that smaller models might otherwise conflate.

Impact of Memory Nodes. We assess the system's robustness by varying the available memory budget from 0% to 100% using Qwen3-235B as the backbone. As shown in Table 4, performance improves progressively as the number of memory nodes increases. This confirms that the breadth of historical experiences directly correlates with the system's ability to generalize to novel problems.

Cross-Model Memory Transferability. A pivotal advantage of DCM-Agent lies in the architectural decoupling of memory construction from inference, enabling flexible cross-model synergy. As shown in Table 5, we observe a compelling "knowledge inheritance" effect: structural priors generated by superior models significantly elevate the performance of smaller counterparts. However, performance eventually declines as memory-generating LLMs become exceptionally powerful; we hypothesize that the resulting high-density clus-

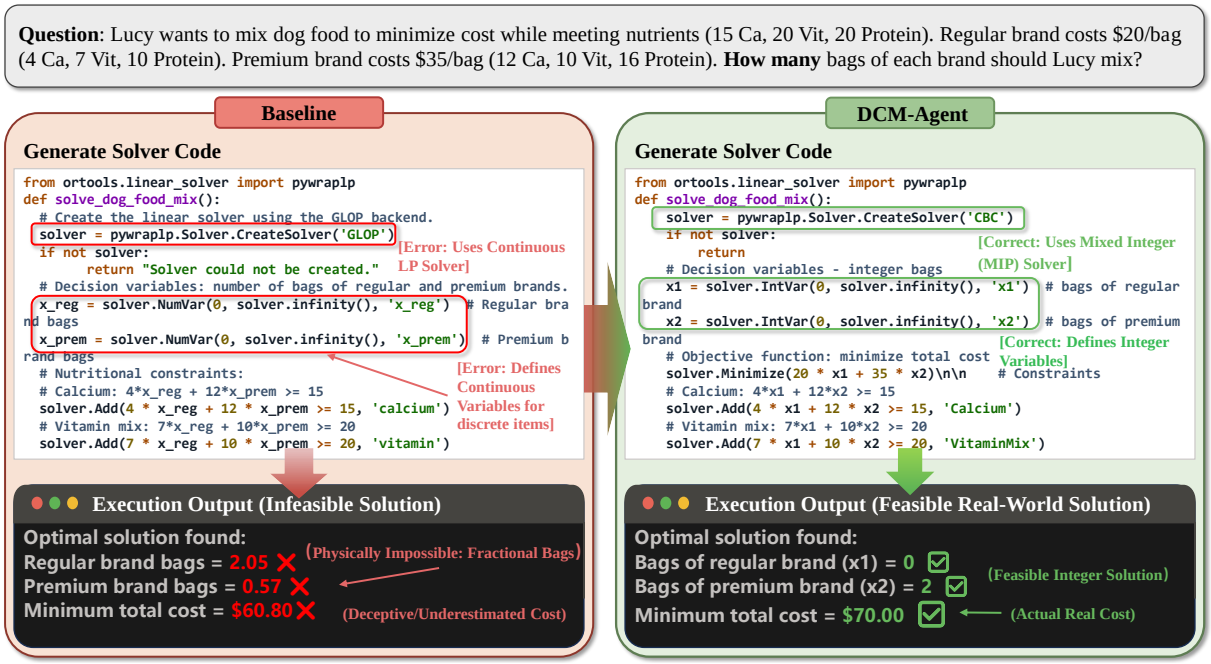

Figure 6: Comparison between the baseline and DCM-Agent on a discrete optimization task. DCM-Agent correctly identifies integrality constraints, whereas the baseline produces a physically infeasible fractional solution.

ter information exceeds the processing capacity of smaller models. This confirms that while weaker models can inherit superior structural reasoning from stronger ones, the complexity remains within the target model's processing threshold.

4.4 Ablation Studies

To assess the impact of DCM-Agent's components, we conduct ablation studies using Qwen3-235B by selectively removing the Modeling and Coding Clusters. As illustrated in Figure 5, the complete framework consistently yields the highest accuracy, validating the synergy of the dual-cluster mechanism. We observe that the removal of Modeling Clusters results in a sharper performance drop compared to removing Coding Clusters (e.g., the blue bars are consistently lower than the orange ones). This result confirms that while executable code guidance is beneficial, decoupling and retrieving precise mathematical logic is the decisive factor for solving complex optimization problems.

4.5 Parameter Analysis

We systematically evaluate DCM-Agent's sensitivity to retrieval size (K), update threshold (N), and planning candidates (M), summarized in Table 6. Both K and N exhibit a distinct bell-shaped trend, peaking at $K = 3$ and $N = 5$. Lower values suffer from insufficient context or unstable generalization (overfitting to isolated samples), while higher values are hampered by excessive input context length or delayed knowledge consolidation. In contrast, increasing M yields consistent monotonic gains by broadening the search space. However, given the marginal improvement at $M = 5$ relative to the incurred computational overhead, we adopt $M = 3$ to ensure the optimal cost-effectiveness.

4.6 Case Study

As shown in Figure 6, DCM-Agent identifies implicit integrality constraints (e.g., discrete bag quantities) and selects an appropriate Mixed-Integer Programming (MIP) solver. Conversely, the baseline overlooks these constraints due to semantic misalignment, treating the problem as a continuous linear task. While the baseline reports a nominally lower cost ($60.80 vs. $70.00), its reliance on fractional quantities results in an infeasible solution. This "deceptive optimality" underscores that numerical performance is secondary to the foundational correctness of the modeling logic, such as the proper distinction between `IntVar` and `NumVar`.

5 Conclusion

We presented the novel Dual-Cluster Memory Agent (DCM-Agent), which addresses structural ambiguity in optimization by decoupling abstract modeling logic from concrete code implementation. By leveraging Dual-Cluster Organization and Evolutionary Experience Stratification, DCM-

Agent provides precise algorithm-specific guidance through stratified historical insights. Empirical results across seven benchmarks confirm that DCM-Agent achieves consistent state-of-the-art accuracy across various model scales while not introducing too much computational overhead. This framework effectively bridges the critical gap between mathematical formulation and executable solver code, significantly improving overall solution robustness. DCM-Agent thus offers a scalable and efficient approach to navigating the highly complex decision space of automated optimization problem solving.

6 Acknowledgements

This work was supported by Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (JYB2025XDXM116), National Natural Science Foundation of China (No. 62137002, 62293550, 62293553, 62293554, 62437002, 62477036, 62477037, 62192781), the Shaanxi Provincial Social Science Foundation Project (No. 2024P041), the Youth Innovation Team of Shaanxi Universities "Multi-modal Data Mining and Fusion", and Xi'an Jiaotong University City College Research Project (No. 2024Y01).

7 Limitation

Despite the superior performance of the Dual-Cluster Memory Agent (DCM-Agent) across various benchmarks and model scales, we identify one primary limitation concerning the framework's initialization process. A notable bottleneck lies in the initialization latency during the memory construction phase. Specifically, distilling structured knowledge requires collecting and classifying historical trajectories into distinct categories and progressively building the bipartite graph. This process incurs a one-time computational overhead that is non-negligible compared to zero-shot prompting. However, it is crucial to emphasize that this represents a "sunk cost": once the Dual-Cluster Memory is fully constructed and stabilized, the framework operates in a plug-and-play manner. The subsequent Memory-Augmented Inference phase is highly efficient, relying on fast embedding retrieval and path planning rather than exhaustive re-generation. This design effectively amortizes the initial construction cost over long-term usage. In future work, we aim to explore online learning mechanisms that enable the memory to evolve dynamically through user interactions, eliminating the need for full reconstruction cycles.

8 Ethical Statement

In developing DCM-Agent, we have rigorously considered the ethical implications of our research, particularly concerning data integrity, privacy, and computational sustainability. Our experiments utilize established, publicly available optimization benchmarks (including NL4Opt, OptiBench, and NLP4LP) and our own collection of 500 questions, which consist solely of mathematical optimization problems and their solutions. We have verified that these datasets do not contain personally identifying information, sensitive data, or offensive content. The problems are purely technical in nature, involving mathematical formulations and programming tasks without any reference to individual identities or potentially harmful content. We strictly adhere to the licensing agreements of these datasets and the large language models employed (e.g., Qwen, DeepSeek, GPT). By demonstrating that smaller models (e.g., 8B parameters) can achieve state-of-the-art performance when augmented with our memory system—often surpassing unaugmented larger models—DCM-Agent offers a pathway to reduce the substantial energy consumption typically associated with running massive foundation models for complex reasoning tasks. This contribution aligns with the broader goal of developing more sustainable and accessible AI systems.

References

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. 2024. Optimus: Scalable optimization modeling with (mi) lp solvers and large language models. In *Forty-first International Conference on Machine Learning*.

Nicolás Astorga, Tennison Liu, Yuanzhang Xiao, and Mihaela van der Schaar. 2025. Autoformulation of mathematical optimization models using llms. In *Forty-second International Conference on Machine Learning*.

Mariam Barry, Gaëtan Caillaut, Pierre Halftermeyer, Raheel Qader, Mehdi Mouayad, Fabrice Le Deit, Dimitri Cariolaro, and Joseph Gesnouin. 2025. Graphrag: Leveraging graph-based efficiency to minimize hallucinations in llm-driven rag for finance data. In *Proceedings of the Workshop on Generative AI and Knowledge Graphs (GenAIK)*, pages 54–65.

S Belil, S Kemmoé-Tchomté, and N Tchernev. 2018. Milp-based approach to mid-term production planning of batch manufacturing environment producing

bulk products. *IFAC-PapersOnLine*, 51(11):1689–1694.

Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. 2025a. Solver-informed rl: Grounding large language models for authentic optimization modeling. *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Yuanyao Chen, Rongsheng Chen, Fu Luo, and Zhenkun Wang. 2025b. Improving generalization of neural combinatorial optimization for vehicle routing problems via test-time projection learning. *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. 2025a. Orlm: A customizable framework in training large models for automated optimization modeling. *Operations Research*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2025b. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. 2025c. LLMs for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 2678–2710, Albuquerque, New Mexico. Association for Computational Linguistics.

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, JUN ZHOU, Aimin Zhou, and Yang Yu. 2025a. Llmopt: Learning to define and solve general optimization problems from scratch. In *The Thirteenth International Conference on Learning Representations*.

Xia Jiang, Yaoxin Wu, Minshuo Li, Zhiguang Cao, and Yingqian Zhang. 2025b. Large language models as end-to-end combinatorial optimization solvers. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. 2025a. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

Chunyang Liu, Houzhen Ma, Hengxu Zhang, Xiaohan Shi, and Fang Shi. 2023. A milp-based battery degradation model for economic scheduling of power system. *IEEE Transactions on Sustainable Energy*, 14(2):1000–1009.

Fan Liu, Zherui Yang, Cancheng Liu, Tianrui Song, Xiaofeng Gao, and Hao Liu. 2025b. Mm-agent: Llm as agents for real-world mathematical modeling problem. *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Haoyang Liu, Jie Wang, Yuyang Cai, Xiongwei Han, Yufei Kuang, and Jianye HAO. 2025c. Optitree: Hierarchical thoughts generation with tree search for llm optimization modeling. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. 2025. Optmath: A scalable bidirectional data synthesis framework for optimization modeling. In *Forty-second International Conference on Machine Learning*.

Mark Meerschaert. 2013. *Mathematical modeling*. Academic press.

OpenAI. 2024. Learning to reason with LLMs.

OpenAI. 2025. Gpt-5.1: A smarter, more conversational chatgpt.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, et al. 2023. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 competition track*, pages 189–203. PMLR.

Ankur Sinha, Shobhit Arora, and Dhaval Pujara. 2025. Autoopt: A dataset and a unified framework for automating optimization problem solving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Yang Wang and Kai Li. 2025. Large language models and operations research: A structured survey. *arXiv e-prints*, pages arXiv–2509.

Zhiyuan Wang, Bokui Chen, Yinya Huang, Qingxing Cao, Ming He, Jianping Fan, and Xiaodan Liang. 2025. ORMind: A cognitive-inspired end-to-end reasoning framework for operations research. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 104–131, Vienna, Austria. Association for Computational Linguistics.

Zihao Wang, Anji Liu, Haowei Lin, Jiaqi Li, Xiaojian Ma, and Yitao Liang. 2024. Rat: Retrieval augmented thoughts elicit context-aware reasoning and verification in long-horizon generation. In *NeurIPS 2024 Workshop on Open-World Agents*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, et al. 2023. Chain-of-experts: When llms meet complex operations research problems. In *The twelfth international conference on learning representations*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. 2025b. Optibench meets resocratic: Measure and improve llms for optimization modeling. In *The Thirteenth International Conference on Learning Representations*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Bowen Zhang and Pengcheng Luo. 2025. Or-llm-agent: Automating modeling and solving of operations research optimization problem with reasoning large language model. *arXiv preprint arXiv:2503.10009*.

Lingling Zhang, Xiaojun Chang, Jun Liu, Minnan Luo, Zhihui Li, Lina Yao, and Alex Hauptmann. 2022. Tn-zstad: Transferable network for zero-shot temporal activity detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3848–3861.

Xinyu Zhang, Yuxuan Dong, Yanrui Wu, Jiaxing Huang, Chengyou Jia, Basura Fernando, Mike Zheng Shou, Lingling Zhang, and Jun Liu. 2025a. PhysReason: A comprehensive benchmark towards physics-based reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16593–16615, Vienna, Austria. Association for Computational Linguistics.

Xinyu Zhang, Yuxuan Dong, Lingling Zhang, Chengyou Jia, Zhuohang Dang, Basura Fernando, Jun Liu, and Mike Zheng Shou. 2025b. Cofft: Chain of foresight-focus thought for visual language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Xinyu Zhang, Lingling Zhang, Yanrui Wu, Muye Huang, and Jun Liu. 2025c. Cognitive predictive coding network: Rethinking the generalization in raven's progressive matrices. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 4097–4106.

Xinyu Zhang, Lingling Zhang, Yanrui Wu, Muye Huang, Wenjun Wu, Bo Li, Shaowei Wang, Basura Fernando, and Jun Liu. 2025d. Diagram-driven course questions generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 5995–6010.

Xinyu Zhang, Lingling Zhang, Yanrui Wu, Shaowei Wang, Wenjun Wu, Muye Huang, Qianying Wang, and Jun Liu. 2025e. Memory-enriched thought-by-thought framework for complex diagram question answering. *Computer Vision and Image Understanding*, page 104608.

Jie Zhao and Kang Hao Cheong. 2025. Structure-aware cooperative ensemble evolutionary optimization on combinatorial problems with multimodal large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

A Details of Ai Assistants In Research Or Writing

We used Claude-4.5-Sonnet and Gemini-3.0-Pro to help us write code and polish the paper.

B Detailed Algorithm Descriptions

B.1 Dual-Cluster Memory Construction

Algorithm 1 describes the process of constructing our dual-cluster memory system from historical problem-solution trajectories. The algorithm operates in two phases: node-level construction and cluster-level evolution. In the node-level phase, each historical problem-solution pair is first categorized into three types based on its solving trajectory: Type A (always correct), Type B (recovered), and Type C (persistent failures). We then decompose each solution into separate modeling logic m_i and coding implementation c_i components, generate their semantic embeddings (e_i^m, e_i^c), and extract structured knowledge $\Phi_i = \langle \phi_i^{approach}, \phi_i^{checklist}, \phi_i^{pitfall} \rangle$. The approach knowledge captures solution templates, the checklist defines verification criteria, and the pitfall documents common errors.

In the cluster-level phase, we organize nodes into coherent clusters using a hybrid approach combining embedding similarity with LLM-powered semantic verification. For each node, its modeling component is assigned to a cluster $\mathcal{C}_j^M$ by retrieving

Algorithm 1 Dual-Cluster Memory Construction

Require: Historical problem-solution pairs $\mathcal{D}_{raw}$
Ensure: Modeling clusters $\mathcal{C}^M$, Coding clusters $\mathcal{C}^C$, Bipartite graph $\mathcal{G}$

// Phase 1: Node-Level Construction
for each problem-solution pair $(x_i, y_i) \in \mathcal{D}_{raw}$ **do**
  Classify into Type A/B/C based on solving attempts
  Decompose into modeling logic m_i and coding implementation c_i
  Generate embeddings: $e_i^m \leftarrow \text{Embed}(m_i)$, $e_i^c \leftarrow \text{Embed}(c_i)$
  Extract instance-level knowledge:
    $\phi_i^{approach} \leftarrow \text{Extract}_{\text{approach}}(\text{Type A/B})$
    $\phi_i^{checklist} \leftarrow \text{Extract}_{\text{checklist}}(\text{Type A/B})$
    $\phi_i^{pitfall} \leftarrow \text{Extract}_{\text{pitfall}}(\text{Type B/C})$
  $\Phi_i \leftarrow \langle \phi_i^{approach}, \phi_i^{checklist}, \phi_i^{pitfall} \rangle$
end for
// Phase 2: Cluster-Level Evolution
for each node n_i with (e_i^m, e_i^c, Φ_i) **do**
  // Modeling Cluster Assignment
  $\mathcal{C}_{cand}^M \leftarrow \text{TopK-Retrieve}(e_i^m, \{\mu_k^M\})$ ▷ k candidate clusters
  if $\text{LLM}_{\text{verify}}(m_i, \mathcal{C}_{cand}^M)$ returns match at cluster $\mathcal{C}_j^M$ **then**
    Add n_i to $\mathcal{C}_j^M$
  else
    Create new cluster $\mathcal{C}_{new}^M$ with centroid $\mu_{new}^M = e_i^m$
  end if
  // Coding Cluster Assignment (similar process)
  Assign c_i to coding cluster $\mathcal{C}_k^C$
  // Knowledge Update
  if $|\mathcal{C}_j^M|_{new} \geq N$ **then** ▷ Threshold reached
    $\mathcal{K}_j^M \leftarrow \text{LLM}_{\text{synth}}\left(\mathcal{K}_j^{M(t)} \cup \bigcup_{n \in \text{new}} \Phi_n\right)$
  end if
  // Bipartite Graph Update
  Update edge weight: $w_{jk} \leftarrow w_{jk} + 1$ for $(\mathcal{C}_j^M, \mathcal{C}_k^C)$
end for
  return $\mathcal{C}^M, \mathcal{C}^C, \mathcal{G} = (\mathcal{V}^M, \mathcal{V}^C, \mathcal{E})$

top-k similar clusters and verifying semantic alignment. The coding component undergoes identical clustering independently. When a cluster accumulates N new nodes, we trigger knowledge synthesis where an LLM consolidates instance-level knowledge into generalized cluster-level guidelines. Finally, the natural associations between modeling and coding clusters form a weighted bipartite graph $\mathcal{G}$, where edge weights indicate co-occurrence frequencies and capture proven compatibility between modeling paradigms and coding strategies.

B.2 Memory-Augmented Inference

Algorithm 2 presents our inference procedure for solving novel optimization problems using the constructed dual-cluster memory. The algorithm implements a systematic Generate-Verify-Repair-Backtrack pipeline guided by cluster knowledge.

The inference begins with a dual-retrieval phase. We first encode the new problem into an embedding e_{new} and retrieve the top-K most similar historical instances, capturing fine-grained problem patterns. Complementary to this, we retrieve top-K modeling clusters by comparing e_{new} with cluster centroids, capturing general algorithmic paradigms. We combine both retrieval sources and, for each identified modeling cluster, query the bipartite graph to retrieve top-k compatible coding clusters based on edge weights. This generates a pool of candidate solution paths $\mathcal{P}$, each representing a complete modeling-to-coding pipeline. An LLM selector then ranks these paths by their align-

Algorithm 2 Memory-Augmented Inference

Require: New problem x_{new}, Memory $\mathcal{D}$, Bipartite graph $\mathcal{G}$
Ensure: Solution $\hat{y}$ (modeling $\hat{m}$ and code $\hat{c}$)

// Phase 1: Dual-Retrieval
Encode problem: $e_{new} \leftarrow \text{Embed}(x_{new})$
$\mathcal{H} \leftarrow \arg\max_K \{\text{sim}(e_{new}, e_i) | x_i \in \mathcal{D}\}$ ▷ Instance-Level Retrieval
$\mathcal{S}_{cluster} \leftarrow \arg\max_K \{\text{sim}(e_{new}, \mu_k^M)\}$ ▷ Cluster-Level Retrieval
$\mathcal{R} \leftarrow \{\mathcal{C}^M(x_i) | x_i \in \mathcal{H}\} \cup \mathcal{S}_{cluster}$ ▷ Combine Retrieval Results
// Generate Trajectory Pool via Graph
$\mathcal{P} \leftarrow \emptyset$
for each $\mathcal{C}_i^M \in \mathcal{R}$ **do**
    $\mathcal{N}_k \leftarrow \text{TopK-Neighbors}(\mathcal{C}_i^M, \mathcal{G})$ ▷ Top-k coding clusters
    for each $\mathcal{C}_j^C \in \mathcal{N}_k$ **do**
        Add path $p = (\mathcal{C}_i^M, \mathcal{C}_j^C)$ to $\mathcal{P}$
    end for
end for
// Rank Paths
$\mathcal{Q} \leftarrow \text{TopM}_{p \in \mathcal{P}}(\text{LLM}_{\text{select}}(p | x_{new}))$
// Phase 2: Solving via Generalized Knowledge
for each path $p_t = (\mathcal{C}_i^M, \mathcal{C}_j^C) \in \mathcal{Q}$ **do**
    // Modeling Phase
    $\hat{m}_{raw} \leftarrow \text{LLM}_{\text{gen}}(x_{new} | \mathcal{K}_i^{approach})$
    $\hat{m} \leftarrow \text{LLM}_{\text{verify}}(\hat{m}_{raw} | \mathcal{K}_i^{checklist})$
    // Coding Phase
    $\hat{c}_{raw} \leftarrow \text{LLM}_{\text{gen}}(\hat{m} | \mathcal{K}_j^{approach})$
    $\hat{c} \leftarrow \text{LLM}_{\text{verify}}(\hat{c}_{raw} | \mathcal{K}_j^{checklist})$
    // Execution & Repair
    $(result, error) \leftarrow \mathcal{E}(\hat{c})$ ▷ Execute code
    if $error =$ None **then return** $\hat{y} = (result, \hat{m}, \hat{c})$
    else
        // Repair Attempt
        $\hat{c}_{fixed} \leftarrow \text{LLM}_{\text{fix}}(\hat{c}, error | \mathcal{K}_j^{pitfall}, \mathcal{K}_j^{checklist})$
        $(result, error) \leftarrow \mathcal{E}(\hat{c}_{fixed})$
        if $error =$ None **then return** $\hat{y} = (result, \hat{m}, \hat{c}_{fixed})$
        end if
    end if
    // Backtrack to next path
    Continue to next p_{t+1} in $\mathcal{Q}$
end for
return Failure ▷ All paths exhausted

ment with the new problem, forming a prioritized queue $\mathcal{Q}$.

The solving phase executes paths from $\mathcal{Q}$ sequentially. For each path, we first generate modeling logic $\hat{m}$ guided by the modeling cluster's approach knowledge and verify it against the checklist. Once validated, we generate executable code $\hat{c}$ using the coding cluster's knowledge and similarly verify it. The code is then executed by the solver engine $\mathcal{E}$. On success, we return the solution. On failure, we initiate knowledge-guided repair by providing the repair LLM with the error message, cluster-specific pitfalls, and a verification checklist. This targeted repair dramatically outperforms generic debugging by grounding fixes in paradigm-specific failure modes. If repair attempts fail within the predefined limit (default: 2 attempts), we backtrack to the next path in $\mathcal{Q}$.

C Optimization Solvers and Libraries

Our framework supports multiple optimization solvers to handle diverse problem types.

Gurobi is a state-of-the-art free research solver for linear programming (LP), mixed-integer linear programming (MILP), and quadratic programming (QP), providing exact solutions with provable optimality guarantees. Its branch-and-cut algorithm excels at large-scale combinatorial problems such as facility location, scheduling, and resource allocation.

PuLP is an open-source linear programming modeler offering a unified Python interface to various solvers, including CBC, GLPK, and CPLEX. It is particularly suitable for rapid prototyping and straightforward LP/MILP problems.

OR-Tools is Google's optimization suite featuring constraint programming (CP-SAT) and routing solvers. It handles combinatorial problems with complex logical constraints, such as scheduling with precedence relations, assignment problems, and vehicle routing, supporting both linear and non-linear constraints.

SciPy provides gradient-based methods (L-BFGS-B, SLSQP) and derivative-free algorithms (Nelder-Mead) for continuous optimization. It is effective for non-linear programming problems, including parameter tuning, curve fitting, and engineering design with non-convex objectives.

NetworkX is a graph analysis library providing efficient implementations of classic algorithms like Dijkstra's shortest path, Ford-Fulkerson maximum flow, and minimum spanning tree. It serves as a specialized solver for network optimization problems with clear graph topology, such as transportation and communication networks.

The bipartite graph in our dual-cluster memory learns associations between modeling paradigms and solver choices through historical co-occurrence patterns. For instance, integer linear programs typically pair with Gurobi or OR-Tools clusters, while continuous non-linear problems align with SciPy clusters. During inference, the framework automatically selects appropriate solvers by querying this learned graph (Algorithm 2), and supports fallback to alternative solvers if the primary choice fails, ensuring robustness across different settings.

D Details of Human Annotators

For the collection, annotation, and verification of the 500 optimization problems in our dataset, we engage Ph.D. candidates and Master's students with expertise in Operations Research and Applied Mathematics, who are also co-authors of this paper. All annotators possess strong academic backgrounds, ensuring their qualifications to accurately formulate problems, verify solution correctness, and maintain the technical precision of domain-specific terminology and mathematical notations. Since the annotators are co-authors involved in this study, no formal external recruitment process or monetary compensation is required, and they are fully informed of the data collection and usage protocols. The annotation process focuses exclusively on creating and evaluating optimization problems, mathematical formulations, and solution approaches, without involving the collection of any personally identifying information or exposing annotators to potential risks. As this research centers on the development and analysis of mathematical optimization content rather than involving external human subjects or sensitive data, it is determined to be exempt from formal institutional review board approval.