CO-Bench: Benchmarking Language Model Agents in Algorithm Search for Combinatorial Optimization

Weiwei Sun*, **Shengyu Feng***, **Shanda Li, Yiming Yang**

Carnegie Mellon University
{weiweis,shengyuf,shandal,yiming}@cs.cmu.edu

Abstract

Although LLM-based agents have attracted significant attention in domains such as software engineering and machine learning research, their role in advancing combinatorial optimization (CO) remains relatively underexplored. This gap underscores the need for a deeper understanding of their potential in tackling structured, constraint-intensive problems—a pursuit currently limited by the absence of comprehensive benchmarks for systematic investigation. To address this, we introduce CO-Bench, a benchmark suite featuring 36 real-world CO problems drawn from a broad range of domains and complexity levels. CO-Bench includes structured problem formulations and curated data to support rigorous investigation of LLM agents. We evaluate multiple agentic frameworks against established human-designed algorithms, revealing the strengths and limitations of existing LLM agents and identifying promising directions for future research.

Code — https://github.com/sunnweiwei/CO-Bench
Data — https://sunnweiwei.github.io/co-bench
Extended version — https://arxiv.org/pdf/2504.04310

Introduction

Combinatorial Optimization (CO) is a foundational problem class in computer science and operation research, focused on finding optimal solutions in discrete, structured, and constraint-rich domains. It underpins a wide range of real-world applications, including logistics (Vogiatzis and Pardalos 2013), production planning (Crama 1997), bioinformatics (Gusfield 1997), etc. Many CO problems are computationally intractable and classified as NP-hard, making exact solutions impractical at scale. As a result, developing effective algorithms often demands significant domain expertise and manual effort—posing a long-standing challenge in both academic research and industrial applications.

Recent advances in Large Language Models (LLMs) (OpenAI 2024b; DeepSeek-AI 2025) have positioned LLM-based agents as increasingly promising tools for a variety of prediction and decision-making tasks (Jimenez et al. 2023; Chan et al. 2024). In particular, there is growing interest in applying LLMs to CO problems. Initial investigations have largely focused on solution correctness, evaluated on small-scale test instances (Ramamonjison et al. 2022; Yang et al. 2025; Xiao et al. 2024), and are often geared towards solving problems posed by general users. More recent works have begun to explore autonomous LLM agents capable of conducting research and designing more efficient algorithms for complex scientific and industrial challenges. For example, FunSearch (Romera-Paredes et al. 2023) combines LLM prompting with evolutionary search to discover heuristics that outperform human-designed counterparts in the Cap Set and Bin Packing problems. Subsequent methods (Liu et al. 2024; Ye et al. 2024; Team 2025) further improve computational efficiency and broaden applicability to domains such as routing and scheduling.

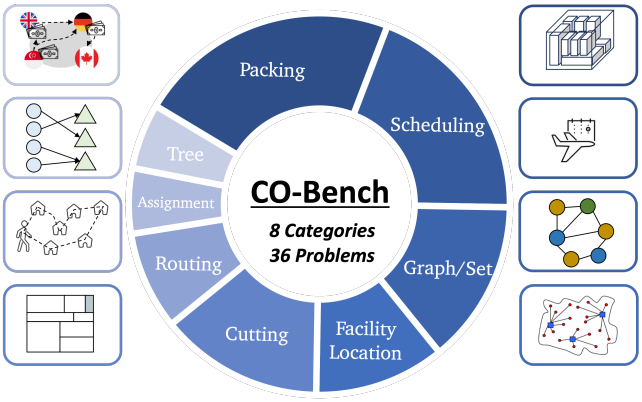

Figure 1: Overview of CO-Bench. CO-Bench includes 36 problems from 8 categories, and aims to evaluate LLM agents' ability to develop effective and efficient algorithms for solving real-world combinatorial optimization problems.

Despite these advancements, most existing efforts focus on narrow components (e.g., priority functions) within established algorithms, across a limited set of tasks (typically 4-7 problems), and often rely on heavily handcrafted, problem-specific prompts and templates (Romera-Paredes et al. 2023; Ye et al. 2024). Furthermore, there remains a lack of systematic evaluation of how these agents perform across a broader and more diverse collection of real-world CO problems.

To address this gap, we introduce **CO-Bench**, a comprehensive benchmark designed to evaluate LLM agents in the

*These authors contributed equally.

context of efficient CO algorithm development. CO-Bench comprises real-world CO problems spanning a wide range of domains and complexities. Figure 1 illustrates the problem categories and examples, while Table 1 compares CO-Bench with existing CO benchmarks. Compared to prior benchmarks, CO-Bench offers broader problem coverage, and emphasizes end-to-end evaluation of LLM-based research agents, focusing on their ability to design efficient, potentially novel algorithms from *abstract problem descriptions*. This design enables reproducible and scalable evaluation of agent performance, including comparisons with human-designed classical CO solver under equivalent time constraints. By doing so, CO-Bench introduces new challenges for LLM agent development, such as the discovery of algorithms that extend beyond current human knowledge of CO.

Using CO-Bench, we benchmark 15 LLMs and 9 agentic frameworks, comparing their performances against both human-designed classical algorithms and the best-known solutions reported in the literature. Our results show that reasoning models (e.g., o3-mini and Claude-3.7-sonnet) consistently outperform standard no-reasoning LLMs. When integrated into agentic frameworks like FunSearch, LLMs further improve through trial-and-error exploration. Notably, on 25 problems, LLM-generated algorithms outperformed classical solvers, and on 3 problems, they surpassed the best-known solutions. However, our analysis also reveals current limitations, such as limited algorithmic novelty and insufficient handling of feasibility constraints. These findings highlight both the promise and challenges of LLM-driven research in CO and suggest key directions for advancing autonomous algorithm design.

In summary, this paper makes the following contributions: (i) We introduce CO-Bench, the first comprehensive benchmark to evaluate the capability of LLMs to develop algorithms for diverse and challenging real-world CO problems (ii) We benchmark 15 LLMs and 9 agentic frameworks, analyzing their performance relative to expert-designed pipelines. Our results highlight the strengths of agent-generated algorithms, while also revealing limitations in planning, feasibility checking, and the generation of efficient solution.

Dataset	Algorithm Dev	Problem Num	Instance Num	Largest Variables
NPHardEval	✗	9	900	24
NL4OPT	✗	5	289	3
OptiBench	✗	4	605	18
ComplexOR	✗	20	20	9
ReEvo	✓	7	597	1,000
CO-Bench	✓	**36**	**6,482**	**11,000**

Table 1: Data statistics for CO-Bench and related CO benchmarks (Fan et al. 2024; Ramamonjison et al. 2022; Yang et al. 2025; Xiao et al. 2024; Ye et al. 2024), including the indicator for algorithm development support, the number of problem types (based on objective functions), the number of test-set problem instances, and the largest number of test-set variables (e.g., the number of nodes in the largest graph).

Preliminary

Combinatorial Optimization

For each CO problem c (for example, Traveling salesman problem), we follow Papadimitriou and Steiglitz (1982) to formulate it as a constrained optimization problem in the discrete space. Consider an instance p, the optimization problem could be expressed as

$$\min_{x \in S_c(p)} f_c(x; p) + g_c(x; p), \tag{1}$$

where $S_c(p)$ represents the solution space, e.g., $\mathbf{Z}^m \times \mathbb{R}^n$ for d discrete variables and n continuous variables, $f_c(x; p)$ corresponds to the objective function, and $g_c(x; p)$ stands for the constraint violation, which is 0 for feasible solutions and $+\infty$ otherwise. To avoid the clutter, we simply denote $h_c(x; p) = f_c(x; p) + g_c(x; p)$ in the following text and omit c if the context is clear.

Given an algorithm set $\mathcal{A}$ and a problem instance distribution D, the algorithm search problem is defined as

$$\min_{A \in \mathcal{A}} \mathbb{E}_{p \sim D, x \sim A(p)}[h(x; p)]. \tag{2}$$

In contrast to previous neural CO solvers (Bengio, Lodi, and Prouvost 2020) that directly parameterize A with a neural network, we focus on symbolic searching space where $\mathcal{A}$ consists of all algorithms that could be represented by a Python Program, with a maximum number of d tokens, where d is typically decided by the output length limit of an LLM. Considering the popularity of randomized algorithms (Motwani and Raghavan 2013) for CO, we treat the output of an algorithm $A(p)$ as a distribution of solutions, while deterministic algorithms would correspond to the point distributions.

The main endeavor of this work is focused on the shaping of the algorithm set $\mathcal{A}$, the curation of the data distribution D and the definition of h on our collected CO problems.

LLM Agents

Given a CO problem c, a candidate algorithm could be generated by an LLM as

$$A \sim M(\text{textify}(c); \theta), \tag{3}$$

where M denotes an LLM with parameters θ. However, one-time generation usually leads to inexecutable code or suboptimal algorithms (Madaan et al. 2023), and *agentic frameworks* address this by enabling iterative refinement through structured interactions with external tools (e.g., a coding environment). Formally, an agent performs reasoning-action iterations (Yao et al. 2022):

$$r_{t+1} \sim M(\text{textify}_r(c, A_t, H_t); \theta), \tag{4}$$

$$a_{t+1} \sim M(\text{textify}_a(r_{t+1}, H_t); \theta), \tag{5}$$

where r_t is the reasoning step, a_t is the action step (e.g., executing code, evaluating results), and $H_t = (r_i, a_i, \text{result}(a_i))_{i=1}^{t-1}$ maintains the interaction history. Thus, an *LLM agent* is formally defined as an LLM $M(\cdot; \theta)$ guided by a structured workflow specifying iterative external interactions to enhance its outputs.

CO-Bench

We introduce CO-Bench, a comprehensive benchmark designed to evaluate the algorithm development ability of LLM agents on combinatorial optimization (CO) problems. The benchmark consists of 36 problems mainly sourced from OR-Library (Beasley 1990), an established archive containing datasets accumulated by researchers across over 30 years of operations research. These problems span a wide range of realistic CO challenges in academia and industrial applications.

Data Curation

Problem Selection We first perform rigorous filtering and cleaning, and select 36 CO problems that cover diverse domains and complexities, including:

- *Packing problems:* Bin packing (Falkenauer 1996), Multi-Demand Multidimensional Knapsack problem (Cappanera and Trubian 2001), Multidimensional knapsack problem (Petersen 1967), Container loading (Bischoff and Ratcliff 1995; Ivancic 1988), Container loading with weight restrictions (Ratcliff and Bischoff 1998; Bischoff 2006), Packing unequal circles (López and Beasley 2016), Packing unequal rectangles and squares number / area (López and Beasley 2018).
- *Cutting problems:* Assortment problem (Beasley 1985a), Constrained / unconstrained guillotine cutting (Christofides and Whitlock 1977; Beasley 1985b), Constrained non-guillotine cutting (Beasley 1985c, 2004).
- *Facility location problems:* Capacitated / Uncapacitated warehouse location (Beasley 1988, 1993), Capacitated / Uncapacitated p-median problem (Beasley 1985d; Osman and Christofides 1994).
- *Scheduling problems:* Aircraft landing (Beasley et al. 2000, 2004), Crew scheduling (Beasley and Cao 1996), Common due date scheduling (Biskup and Feldmann 2001), Flow shop scheduling (Taillard 1993), Hybrid Reentrant Shop Scheduling (Chakhlevitch and Glass 2009), Job shop scheduling (Taillard 1993), Open shop scheduling (Taillard 1993).
- *Routing problems:* Traveling salesman problem (Laporte 1992), Period vehicle routing problem (Christofides and Beasley 1984), Resource constrained shortest path (Beasley and Christofides 1989).
- *Assignment problems:* Constrained / unconstrained assignment (Osman 1995; and 1990).
- *Tree problems:* Euclidean Steiner (Beasley 1992), Corporate structuring (Anken and Beasley 2012)
- *Graph and set problems:* Maximal Independent Set (Erdos and Rényi 1984), Graph colouring (Fleurent and Ferland 1996), Equitable partitioning (Mingers and O'Brien 1995), Set partitioning (Chu and Beasley 1998), Set covering (Beasley and Jörnsten 1992).

Data Annotation For each problem, we manually annotate the following components: (1) *Problem description*: a formal definition of the optimization problem in natural language, accompanied by a clearly specified `solve` function as the starter code; (2) *Data loading function*: a `load_data` function to load and preprocess raw data from the test files; (3) *Evaluation function*: an `eval_func` function that rigorously and robustly evaluates the quality of a solution. Additionally, each problem comprises a *development* set and a *test* set, each containing several problem instances.

Evaluation Framework We develop a rigorous and efficient evaluation framework to assess the performance of LLM agents in simulated, time-constrained competition scenarios (Chan et al. 2024). Specifically, LLM agents operate within a sandbox environment with access to a Linux machine. For each problem, agents are provided with a problem description, development datasets, and an API endpoint for submitting their solutions (i.e. codebases) to receive evaluation feedback. An independent evaluation system, which is protected by built-in safeguards, scores the submitted solutions on the development set in parallel. After a limited number of research steps, the agent submits its final solution for evaluation on the test set. During the agent development process, both `eval_func` and test data are invisible. Figure 2 shows the evaluation pipeline in CO-Bench.

Designing Classical Solver Baselines To investigate how existing LLM agents perform compared to classical solvers, we establish a *classical solver* baseline. Specifically, the authors of this paper—who have extensive experience in related areas and are familiar with the problems in CO-Bench—spent approximately 30 minutes per problem testing and selecting the most effective classical solvers (e.g., LKH for TSP, CPLEX for scheduling, Gurobi for MIS) and tuning their hyperparameters on the development set. This process ensures that the classical solver baseline is well-tuned and competitive for each problem in CO-Bench.

Evaluation Metrics

Avg Score The main evaluation metric is similar to the *Primal Gap* (Berthold 2006), defined as the normalized score of the primal bound $h(x;p)$ against a pre-computed optimal (or best-known) objective value h_p^*:

$$s(x,p) = \frac{\min\{|h(x,p)|, |h_p^*|\}}{\max\{|h(x,p)|, |h_p^*|\}}, \tag{6}$$

A higher value indicates better performance and a score of 1 signifies the performance identical to the optimal or best-known solution. Program errors or infeasible solutions lead to a score of 0.0. The score of a solver on a given problem is computed by averaging its scores across all test instances. The overall benchmark score is then obtained by averaging these problem-level scores across all 36 problems.

Valid Solution We compute the percentage of problems for which the generated code is correct on all test instances. Any raised error–such as constraint violation or timeout–is treated as an invalid signal. If any test instance for a given problem results in an invalid signal, the entire solution for that problem is considered invalid, even if it produces valid results on other test instances.

Above Classical Given the performance of classical solver, we calculate the portion of problems where the model outperforms the classical solver baseline.

Survival Rate The survival rate measures that, for each problem, the percentage of test instances where the model's solution is above 99% of the reference score (reported optimal or best-known solution from literature). This serve as a

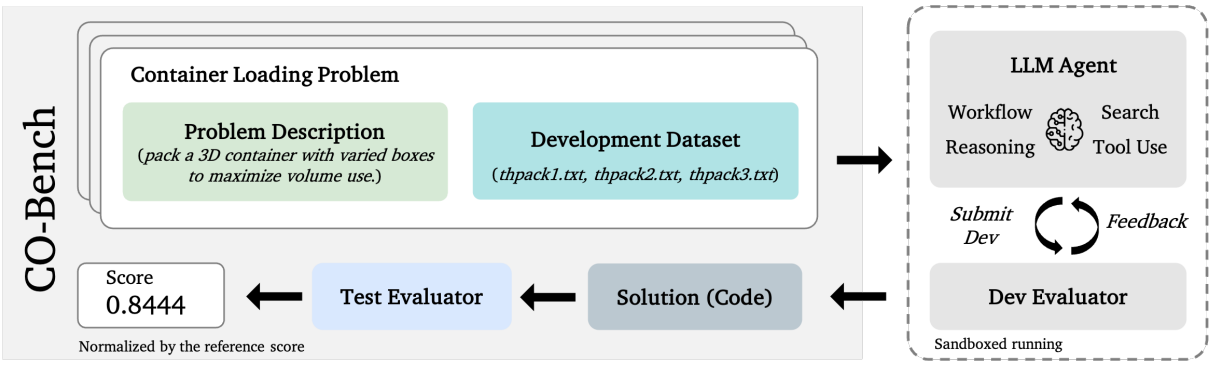

Figure 2: CO-Bench is an evaluation environment for AI agents. Each problem has an associated description and a development dataset. Following the setup in Chan et al. (2024), the agent-generated code implements an algorithm design, which is further graded and compared against the best-known solution and human expert solution.

challenge metric as the model can only get credit when it is very close or better than previous-best algorithm.

Experimental Setup

Benchmarked Methods

On CO-Bench, we evaluate various LLMs combined with different agentic frameworks, and compare them with existing human-designed CO solvers.

LLMs We conduct experiments on 5 open-source models and 10 proprietary models. These include instruction-tuned models such as *Llama-3.3-70B-Instruct* (Meta 2024), *Qwen-2.5-Code-32B-Instruct* (Hui et al. 2024), *DeepSeek-V3* (DeepSeek-AI 2024), and *GPT-4o* (OpenAI 2024a), as well as frontier reasoning models, including *o3-mini* (OpenAI 2025), *Claude-3.7-Sonnet-Thinking* (Anthropic 2025), *DeepSeek-R1* (DeepSeek-AI 2025), *Grok-3-Thinking* (xAI 2025), *QwQ-32B* (Qwen 2025), and *Gemini 2.5 Pro* (DeepMind 2025).

Agentic frameworks For the aforementioned LLMs, we apply various agentic frameworks to evaluate their performance across different strategies. These range from simple approaches, such as direct generation, to more sophisticated frameworks that augment LLM with additional tools, workflows, and test-time compute:

- *Direct Answer:* The simplest approach, where the LLM directly generates a solution to the combinatorial optimization problem without further refinement.
- *BestOfN Sampling (Chen and et al. 2021):* Generate N candidate solutions, evaluate each on a development set, and select the solution with the best performance.
- *Chain of Experts (Xiao et al. 2024):* A multi-agent prompting framework where agents of different roles cooperate to debug and deliver one solution.
- *Greedy Refinement (Shinn et al. 2023; Madaan et al. 2023):* Iteratively prompt the LLM to refine the current best solution based on the evaluation results of the development set, repeating this refinement process for N steps.
- *FunSearch (Romera-Paredes et al. 2023):* Prompt the LLM to either draft a new solution or refine an existing one, followed by employing an evolutionary algorithm to iteratively select and improve candidate solutions.
- *EoH (Liu et al. 2024):* Evolve both thoughts and codes in an evolutionary search framework for generating high-performance heuristics.
- *AIDE (Jiang et al. 2025):* A representative method for machine learning engineering tasks, which stores existing solutions in a tree structure and selectively prompts the LLM to draft new solutions, debug or improve previously stored solutions.
- *ReEvo (Ye et al. 2024):* A recent evolutionary algorithm that incorporates short-term and long-term reflection modules, as well as a multi-agentic framework.
- *MSTC-AHD (Zheng et al. 2025):* A Monte Carlo Tree Search (MCTS)-based agentic pipeline that organizes all LLM-generated heuristics in a tree structure and uses the MCTS algorithm with progressive widening technique to guide the evolution of heuristics.

Implementation Details

For benchmark evaluation, we limit the solving time of each test instance to 10 seconds on a single CPU, such that the exact solving of the problem (achieving the optimal solution) is impossible on most test instances. Test instances that result in a timeout or error receive a score of 0.

For agent implementation, we use o3-mini-medium as the default base model. Since the original implementations of these agents may use different evaluation setups, we adapt their approaches to our benchmark setting (i.e., end-to-end algorithm search) by adjusting the prompts and tools. For all agents, we set the number of iteration steps to 64. In each step, the agent generates a code block as a candidate algorithm and obtains its evaluation score on the development set. After 64 iterations, the agent produces 64 candidate algorithms, from which the best-performing solution on the development set is selected for final benchmark evaluation. All evaluations are conducted on a single CPU core of a dual AMD EPYC 7313 16-Core processor.

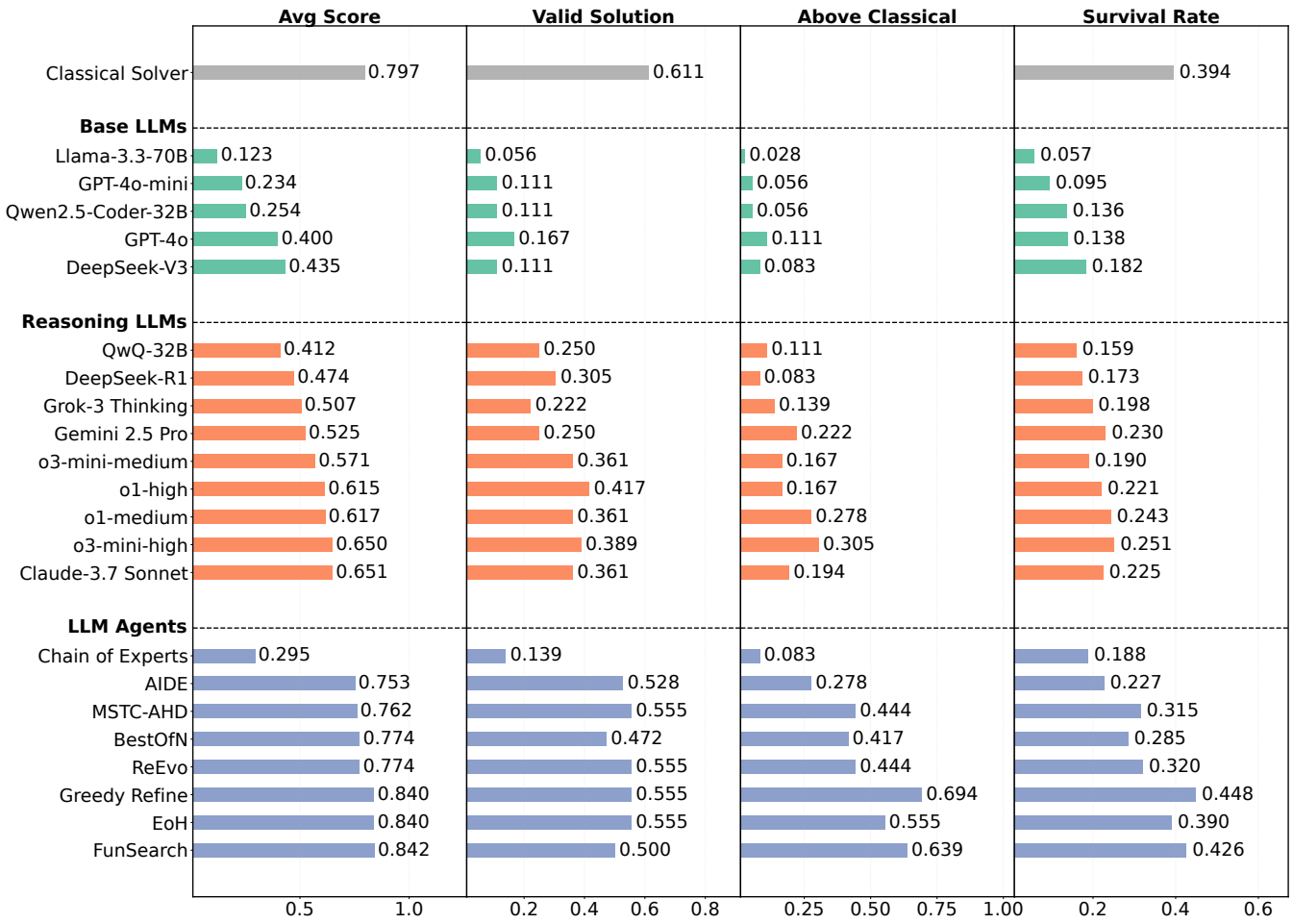

Figure 3: **Overall Performance.** LLM Agents are all based on `o3-mini-medium`. *Avg Score* refers to the average normalized objective scores across all problems. *Valid Solution* indicates the percentage of test-set problems for which the solutions are feasible. *Above Classical* represents the percentage of test instances where the model outperforms the classical solver baseline. *Survival Rate* measures the percentage of test instances where the model's score exceeds 99% of the reference score.

Main Results

Figure 3 presents the results of LLMs and agents on the test set. We highlight the following key findings.

Direct generation performance is limited. LLMs show significantly lower average scores compared to the classical solver. They often fail to generate valid solutions (i.e., bug-free code that satisfies all constraints within the time limit), rarely outperform the classical solver on individual instances, and often fail to produce optimal solutions. Reasoning-capable models tend to perform better than non-reasoning ones. The best-performing LLM for one-shot generation is *Claude-3.7 Sonnet*, with an average score of 0.65.

Agentic systems substantially improve LLM performance. Compared to direct generation, the agentic pipeline achieves considerably higher scores across all metrics. Among the evaluated frameworks, FunSearch attains the highest average score of 0.842, outperforming the classical solver (0.797). It also surpasses the solver on over half the test instances (see "Above Classical" score) and achieves a higher survival rate. These results highlight the effectiveness of LLM-based agents in solving CO problems.

Agent performance varies widely. Some advanced agentic frameworks, such as AIDE, underperform compared to simpler strategies like BestOfN on most metrics, though they show higher valid solution rates—possibly due to their debugging capabilities. This indicates that current planning mechanisms in agents are still underdeveloped and may not reliably outperform random sampling.

Valid solution rates still lag behind classical solvers. According to the *Valid Solution* metric, the best-performing agents achieve a success rate of 0.555—lower than that of the classical solver (0.611). This suggests that current agents often struggle with solution feasibility and reliability.

Agents Error Analysis

To investigate why the agents' valid solution scores are low, Figure 4 shows the types of errors among invalid solutions for five agents. We observe that code errors (i.e., bugs that pre-

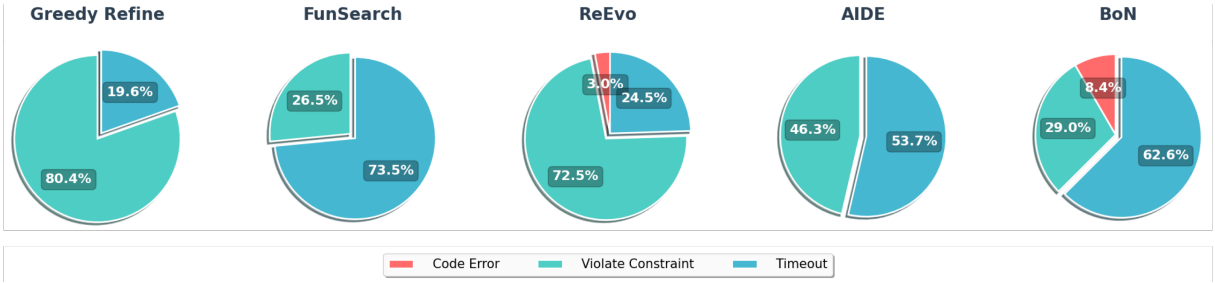

Figure 4: **Agents Error Analysis.** Distribution of three types of errors among invalid solutions for five agents.

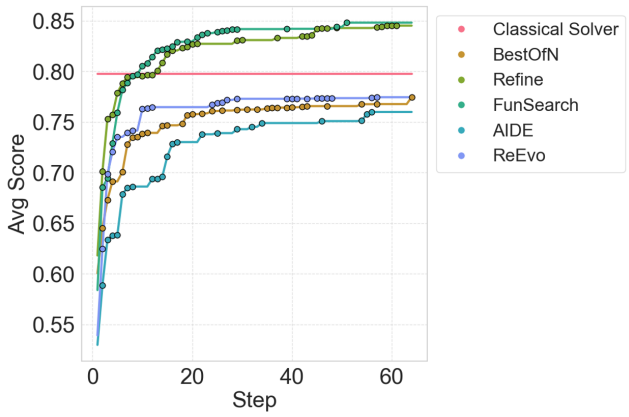

Figure 5: **Avg Score** vs. the number of iteration steps (in total 64 steps) during the algorithm development.

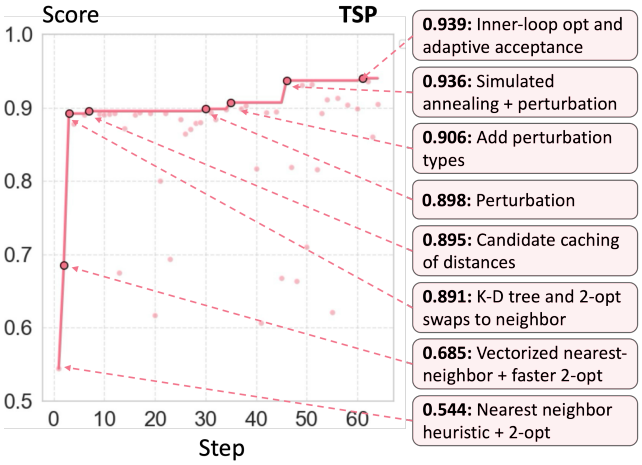

Figure 6: Trajectory of algorithm development for Greedy Refinement on TSP over 64 steps. The curve and highlighted dots indicate the best-ever score and the steps where improvements occurred. The algorithmic ideas behind each improvement step are summarized in corresponding boxes.

vent compilation) are the least frequent issue. The dominant error type varies across agents: Greedy Refine and ReEvo exhibit more constraint violations, while FunSearch, AIDE, and BoN encounter more timeout errors. This highlights agents' limitations in satisfying constraints and generating efficient algorithms within time limits.

Performance over Iteration Steps

Figure 5 illustrates the performance of several representative LLM agents across different iteration steps. At each step, the agent generates a new algorithm and receives evaluation results on the development set. We also include the performance of the classical solver baseline for comparison.

All agents exhibit the ability to improve their performance with more iteration steps. FunSearch consistently achieves the best results, reaching a score of 0.8423 and converging after around 50 steps. Notably, both FunSearch and Refine discover algorithms that outperform the classical solver within approximately 10 steps. However, performance tends to saturate after 30 steps, with further search yielding diminishing returns. Enabling more consistent improvements under longer search budgets presents an interesting future direction.

Figure 6 shows an example trajectory of algorithm development by Greedy Refinement (o3-mini) on TSP over multiple search steps. In the early stages, the agent enhances code efficiency by adopting vectorized data structures and utilizing a K-D tree. It then increases the number of search iterations and introduces perturbations to escape local optima. Finally, the agent integrates simulated annealing to balance exploration and exploitation and applies adaptive heuristics for different instance sizes. This example demonstrates that LLMs excel in applying established techniques to improve efficiency and implementation quality, but failing at algorithmic novelty. See Appendix for more examples.

Comparison to Neural Solvers

Table 2 compares the performance of agents with representative neural solvers on TSP and MIS, two well-studied CO problems. We include DIMES (Qiu, Sun, and Yang 2022), DIFUSCO (Sun and Yang 2023), and T2T (Li et al. 2023) as neural baselines. For the method with multiple variants, we only include their best results on each dataset. We also consider a hybrid method, LEHD + ReEvo (Ye et al. 2024), which combines the neural solver with LLM-designed heuristics. We report both the objective values (the tour length for TSP and set size for MIS) and the solving time. The results show that agents like Greedy Refine and FunSearch outperform neural solvers on MIS tasks but underperform on TSP. They are generally faster than state-of-the-art neural solvers. We also observe that Greedy Refine performs slightly better

	TSP-1K Len$_\downarrow$/Time	TSP-10K Len$_\downarrow$/Time	ER-Small Size$_\uparrow$/Time	ER-Large Size$_\uparrow$/Time
Gurobi	-/-	-/-	41.4/50.0m	-/-
DIMES	26.4/2.4m	85.8/4.8m	42.1/12.0m	332/12.5m
DIFUSCO	23.5/48.1m	**73.9**/6.7h	41.1/26.6m	-/-
T2T	**23.3**/54.6m	-/-	41.4/29.7m	-/-
LEHD + ReEvo	23.8/-	-/-	-/-	-/-
Greedy Refine	24.4/19.1m	77.7/2.5m	**42.4**/20.1m	354/2.5m
FunSearch	25.3/19.1m	80.2/2.5m	41.7/1.9m	**356**/2.1m

Table 2: Objective values and solving time of different solvers on TSP and MIS, with varying data sizes.

than FunSearch on the two problems.

Solution Analysis

In Figure 7, we plot the percentage of algorithms developed by the Greedy Refinement agent for the 36 CO problems that utilize existing solvers (e.g., code importing `ortools`, `pulp`). The percentages are shown across 64 iteration steps. We observe an increasing trend in the use of existing solvers in the agent's solutions. After 64 iterations, the final usage rate reaches 25% (i.e., solutions for 9 problems use existing solvers). The solvers used throughout all steps and problems are limited to three: `ortools`, `pulp`, and `scipy`.

This suggests that while existing LLM agents are capable of developing algorithms without relying on existing solvers for most problems, there is a growing tendency to do so over time. Moreover, the solvers used are basic general-purpose tools rather than state-of-the-art solvers specifically designed for each problem (e.g., LKH for TSP), indicating that the agent lacks the necessary knowledge to select the best-performing solver.

Related Work

Automatic Algorithm Search for CO

Automating algorithm search for combinatorial optimization (CO) has emerged as a significant research direction in the machine learning community. Traditional machine learning solvers primarily parameterize CO algorithms as trainable neural networks (Bengio, Lodi, and Prouvost 2020; Cappart et al. 2023). Although effective in capturing data distributions, these neural approaches often struggle to generate feasible solutions, necessitating integration with human-designed heuristics such as branch-and-bound (Gasse et al. 2019) and tree search (Böther et al. 2022). To address this limitation, Kuang et al. (2024a,b) propose to decompose CO algorithms into symbolic operators and conduct searches in the symbolic space. However, designing these unit symbolic operators demands substantial human expertise, limiting generalizability and comprehensive coverage of all algorithm types. Recent advances in Large Language Models (LLMs) and LLM-based agents have significantly mitigated this challenge by enabling symbolic searching in programming language formats (Romera-Paredes et al. 2023; Ye et al. 2024; Liu et al. 2024; Li et al. 2025). Building on these developments, CO-Bench aims to extend the success of these methods to more real-world CO problems and facilitate further research in this domain.

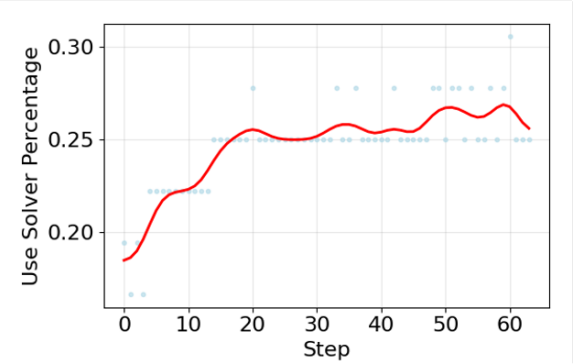

Figure 7: Percentage of algorithms developed by the Greedy Refinement agent that rely on existing solvers (e.g., code importing `ortools`, `pulp`) over 64 iteration steps. We observe an increasing use of existing solvers.

CO Benchmarks for LLMs

Existing CO benchmarks can be roughly classified into two categories. The first type formulates CO problems as question-answering tasks (Fan et al. 2024; Tang et al. 2025). Although LLMs have the potential to solve CO problems via natural language reasoning, their excessive parameter size makes them inefficient CO solvers in general. Therefore, the second type of benchmarks evaluates the tool-using ability of LLMs, e.g., calling an existing CO solver, to address CO problems (Xiao et al. 2024; Ahmaditeshnizi, Gao, and Udell 2024; Yang et al. 2025). However, these benchmarks only evaluate the correctness of the generated algorithm on small-scale CO problems, whose problem parameters could be fully expressed in natural language. In contrast, CO-Bench targets scientific and industrial challenges, emphasizing the evaluation of algorithm efficiency on diverse, large-scale CO instances. This results in a more demanding benchmark, well-suited for assessing powerful reasoning models and agents.

Conclusion

This work introduces CO-Bench, the first benchmark designed to evaluate the ability of LLMs in the search of combinatorial optimization (CO) algorithms. Our systematic evaluation reveals that reasoning-focused LLMs, especially when paired with agentic frameworks, can automatically discover effective algorithms that rival or surpass the classical solvers designed by human experts, with competitive searching time across a wide range of real-world CO problems. At the same time, we identify key limitations of current LLM agents, such as their tendency to struggle with understanding and enforcing problem constraints, maintaining reliability across instances, and exhibiting genuine algorithmic novelty. These shortcomings highlight the need for future research to enhance agents' problem comprehension, constraint-aware reasoning, and search strategies in CO tasks, ultimately enabling more robust, efficient, and autonomous scientific discovery.

References

Ahmaditeshnizi, A.; Gao, W.; and Udell, M. 2024. OptiMUS: Scalable Optimization Modeling with (MI)LP Solvers and Large Language Models. In *ICML 2024*.

and, J. E. B. 1990. Linear Programming on Cray Supercomputers. *Journal of the Operational Research Society*.

Anken, F.; and Beasley, J. E. 2012. Corporate structure optimisation for multinational companies. *Omega*.

Anthropic. 2025. Claude Sonnet.

Beasley, J. E. 1985a. An algorithm for the two-dimensional assortment problem. *EJOR*.

Beasley, J. E. 1985b. Algorithms for Unconstrained Two-Dimensional Guillotine Cutting. *Journal of the Operational Research Society*.

Beasley, J. E. 1985c. An Exact Two-Dimensional Non-Guillotine Cutting Tree Search Procedure. *Operations Research*.

Beasley, J. E. 1985d. A note on solving large p-median problems. *EJOR*.

Beasley, J. E. 1988. An algorithm for solving large capacitated warehouse location problems. *EJOR*.

Beasley, J. E. 1990. OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society*.

Beasley, J. E. 1992. A heuristic for Euclidean and rectilinear Steiner problems. *EJOR*.

Beasley, J. E. 1993. Lagrangean heuristics for location problems. *EJOR*.

Beasley, J. E. 2004. A population heuristic for constrained two-dimensional non-guillotine cutting. *EJOR*.

Beasley, J. E.; and Cao, B. 1996. A tree search algorithm for the crew scheduling problem. *EJOR*.

Beasley, J. E.; and Christofides, N. 1989. An algorithm for the resource constrained shortest path problem. *Networks*.

Beasley, J. E.; and Jörnsten, K. 1992. Enhancing an algorithm for set covering problems. *EJOR*.

Beasley, J. E.; Krishnamoorthy, M.; Sharaiha, Y. M.; and Abramson, D. 2000. Scheduling Aircraft Landings - The Static Case. *Transportation Science*.

Beasley, J. E.; Krishnamoorthy, M.; Sharaiha, Y. M.; and Abramson, D. 2004. Displacement problem and dynamically scheduling aircraft landings. *Journal of the Operational Research Society*.

Bengio, Y.; Lodi, A.; and Prouvost, A. 2020. Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon.

Berthold, T. 2006. *Primal heuristics for mixed integer programs*. Ph.D. thesis, Zuse Institute Berlin (ZIB).

Bischoff, E. E. 2006. Three-dimensional packing of items with limited load bearing strength. *EJOR*.

Bischoff, E. E.; and Ratcliff, M. S. W. 1995. Issues in the development of approaches to container loading. *Omega*.

Biskup, D.; and Feldmann, M. 2001. Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. *Computers & Operations Research*.

Böther, M.; Kißig, O.; Taraz, M.; Cohen, S.; Seidel, K.; and Friedrich, T. 2022. What's Wrong with Deep Learning in Tree Search for Combinatorial Optimization. In *ICLR 2022*.

Cappanera, P.; and Trubian, M. 2001. A Local-Search-Based Heuristic for the Demand-Constrained Multidimensional Knapsack Problem. *INFORMS Journal on Computing*.

Cappart, Q.; ChÃ©telat, D.; Khalil, E. B.; Lodi, A.; Morris, C.; and VeliÄkoviÄ‡, P. 2023. Combinatorial Optimization and Reasoning with Graph Neural Networks. *Journal of Machine Learning Research*.

Chakhlevitch, K.; and Glass, C. A. 2009. Scheduling reentrant jobs on parallel machines with a remote server. *Computers & Operations Research*.

Chan, J. S.; Chowdhury, N.; Jaffe, O.; Aung, J.; Sherburn, D.; Mays, E.; Starace, G.; Liu, K.; Maksin, L.; Patwardhan, T. A.; Weng, L.; and Mkadry, A. 2024. MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering. *ArXiv*.

Chen, M.; and et al. 2021. Evaluating Large Language Models Trained on Code. *ArXiv*.

Christofides, N.; and Beasley, J. E. 1984. The period routing problem. *Networks*.

Christofides, N.; and Whitlock, C. 1977. An Algorithm for Two-Dimensional Cutting Problems. *Operations Research*.

Chu, P. C.; and Beasley, J. E. 1998. Constraint Handling in Genetic Algorithms: The Set Partitioning Problem. *Journal of Heuristics*.

Crama, Y. 1997. Combinatorial optimization models for production scheduling in automated manufacturing systems. *European Journal of Operational Research*.

DeepMind, G. 2025. Flash Thinking: Behind the Scenes of Gemini.

DeepSeek-AI. 2024. DeepSeek-V3 Technical Report. *ArXiv*.

DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *ArXiv*.

Erdos, P. L.; and Rényi, A. 1984. On the evolution of random graphs. *Transactions of the American Mathematical Society*.

Falkenauer, E. 1996. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*.

Fan, L.; Hua, W.; Li, L.; Ling, H.; and Zhang, Y. 2024. NPHardEval: Dynamic Benchmark on Reasoning Ability of Large Language Models via Complexity Classes. In *ACL 2024*.

Fleurent, C.; and Ferland, J. A. 1996. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*.

Gasse, M.; Chételat, D.; Ferroni, N.; Charlin, L.; and Lodi, A. 2019. Exact Combinatorial Optimization with Graph Convolutional Neural Networks. In *NeurIPS 2019*.

Gusfield, D. 1997. Algorithms on stings, trees, and sequences: Computer science and computational biology. *Acm Sigact News*.

Hui, B.; Yang, J.; Cui, Z.; Yang, J.; Liu, D.; Zhang, L.; Liu, T.; Zhang, J.; Yu, B.; Dang, K.; Yang, A.; Men, R.; Huang, F.; Quan, S.; Ren, X.; Ren, X.; Zhou, J.; and Lin, J. 2024. Qwen2.5-Coder Technical Report. *ArXiv*.

Ivancic, N. J. 1988. An integer programming based heuristic approach to the three dimensional packing problem.

Jiang, Z.; Schmidt, D.; Srikanth, D.; Xu, D.; Kaplan, I.; Jacenko, D.; and Wu, Y. 2025. AIDE: AI-Driven Exploration in the Space of Code. *ArXiv*.

Jimenez, C. E.; Yang, J.; Wettig, A.; Yao, S.; Pei, K.; Press, O.; and Narasimhan, K. 2023. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *ArXiv*.

Kuang, Y.; Wang, J.; Liu, H.; Zhu, F.; Li, X.; Zeng, J.; HAO, J.; Li, B.; and Wu, F. 2024a. Rethinking Branching on Exact Combinatorial Optimization Solver: The First Deep Symbolic Discovery Framework. In *ICLR 2024*.

Kuang, Y.; Wang, J.; Zhou, Y.; Li, X.; Zhu, F.; Hao, J.; and Wu, F. 2024b. Towards General Algorithm Discovery for Combinatorial Optimization: Learning Symbolic Branching Policy from Bipartite Graph. In *ICML 2024*.

Laporte, G. 1992. The traveling salesman problem: An overview of exact and approximate algorithms. *EJOR*.

Li, S.; Sun, W.; Li, S.; Talwalkar, A.; and Yang, Y. 2025. Towards Community-Driven Agents for Machine Learning Engineering. *ArXiv*.

Li, Y.; Guo, J.; Wang, R.; and Yan, J. 2023. From Distribution Learning in Training to Gradient Search in Testing for Combinatorial Optimization. In *NeurIPS 2023*.

Liu, F.; Tong, X.; Yuan, M.; Lin, X.; Luo, F.; Wang, Z.; Lu, Z.; and Zhang, Q. 2024. Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model. In *ICML 2024*.

López, C. O.; and Beasley, J. E. 2016. A formulation space search heuristic for packing unequal circles in a fixed size circular container. *EJOR*.

López, C. O.; and Beasley, J. E. 2018. Packing unequal rectangles and squares in a fixed size circular container using formulation space search. *Computers & Operations Research*.

Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; Welleck, S.; Majumder, B. P.; Gupta, S.; Yazdanbakhsh, A.; and Clark, P. 2023. Self-Refine: Iterative Refinement with Self-Feedback. *ArXiv*.

Meta. 2024. The Llama 3 Herd of Models. *ArXiv*.

Mingers, J. C.; and O'Brien, F. A. 1995. Creating student groups with similar characteristics: A heuristic approach. *Omega*.

Motwani, R.; and Raghavan, P. 2013. *Randomized Algorithms*. Cambridge University Press.

OpenAI. 2024a. GPT-4o System Card.

OpenAI. 2024b. OpenAI o1 System Card.

OpenAI. 2025. OpenAI o3-mini System Card.

Osman, I. H. 1995. Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. *OR Spectrum*.

Osman, I. H.; and Christofides, N. 1994. Capacitated clustering problems by hybrid simulated annealing and tabu search. *ITOR*.

Papadimitriou, C.; and Steiglitz, K. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Courier Corporation.

Petersen, C. C. 1967. Computational Experience with Variants of the Balas Algorithm Applied to the Selection of R&D Projects. *Management Science*.

Qiu, R.; Sun, Z.; and Yang, Y. 2022. DIMES: A Differentiable Meta Solver for Combinatorial Optimization Problems. In *NeurIPS 2022*.

Qwen. 2025. QwQ-32B: Embracing the Power of Reinforcement Learning.

Ramamonjison, R.; Yu, T.; Li, R.; Li, H.; Carenini, G.; Ghaddar, B.; He, S.; Mostajabdaveh, M.; Banitalebi-Dehkordi, A.; Zhou, Z.; and Zhang, Y. 2022. NL4Opt Competition: Formulating Optimization Problems Based on Their Natural Language Descriptions. In *NeurIPS 2022*.

Ratcliff, M. S. W.; and Bischoff, E. E. 1998. Allowing for weight considerations in container loading. *OR Spectrum*.

Romera-Paredes, B.; Barekatain, M.; Novikov, A.; Balog, M.; Kumar, M. P.; Dupont, E.; Ruiz, F. J. R.; Ellenberg, J. S.; Wang, P.; Fawzi, O.; Kohli, P.; Fawzi, A.; Grochow, J.; Lodi, A.; Mouret, J.-B.; Ringer, T.; and Yu, T. 2023. Mathematical discoveries from program search with large language models. *Nature*.

Shinn, N.; Cassano, F.; Labash, B.; Gopinath, A.; Narasimhan, K.; and Yao, S. 2023. Reflexion: language agents with verbal reinforcement learning. In *NeurIPS 2023*.

Sun, Z.; and Yang, Y. 2023. DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Optimization. *ArXiv*.

Taillard, E. 1993. Benchmarks for basic scheduling problems. *EJOR*.

Tang, J.; Zhang, Q.; Li, Y.; Chen, N.; and Li, J. 2025. GraphArena: Evaluating and Improving Large Language Models on Graph Computation. In *ICLR 2025*.

Team, G. D. 2025. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *ArXiv*.

Vogiatzis, C.; and Pardalos, P. 2013. *Combinatorial optimization in transportation and logistics networks*. Springer.

xAI. 2025. Grok-3 and the Next Phase of xAI.

Xiao, Z.; Zhang, D.; Wu, Y.; Xu, L.; Wang, Y. J.; Han, X.; Fu, X.; Zhong, T.; Zeng, J.; Song, M.; and Chen, G. 2024. Chain-of-Experts: When LLMs Meet Complex Operations Research Problems. In *ICLR 2024*.

Yang, Z.; Wang, Y.; Huang, Y.; Guo, Z.; Shi, W.; Han, X.; Feng, L.; Song, L.; Liang, X.; and Tang, J. 2025. OptiBench Meets ReSocratic: Measure and Improve LLMs for Optimization Modeling. In *ICLR 2025*.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2022. ReAct: Synergizing Reasoning and Acting in Language Models. *ArXiv*.

Ye, H.; Wang, J.; Cao, Z.; Berto, F.; Hua, C.; Kim, H.; Park, J.; and Song, G. 2024. ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution. In *NeurIPS 2024*.

Zheng, Z.; Xie, Z.; Wang, Z.; and Hooi, B. 2025. Monte Carlo Tree Search for Comprehensive Exploration in LLM-Based Automatic Heuristic Design. *ArXiv*.