arXiv:2506.07972v1 [cs.LG] 9 Jun 2025

HeuriGym: An Agentic Benchmark for LLM-Crafted Heuristics in Combinatorial Optimization

Hongzheng Chen[1*] **Yingheng Wang**[1*] **Yaohui Cai**[1*] **Hins Hu**[1*] **Jiajie Li**[1*]
Shirley Huang[2] **Chenhui Deng**[3] **Rongjian Liang**[3] **Shufeng Kong**[1]
Haoxing Ren[3] **Samitha Samaranayake**[1] **Carla P. Gomes**[1] **Zhiru Zhang**[1]

[1] Cornell University [2] Harvard University [3] NVIDIA Corporation

{hzchen,yingheng}@cs.cornell.edu, {yc2632,zh223,jl4257}@cornell.edu

Abstract

While Large Language Models (LLMs) have demonstrated significant advancements in reasoning and agent-based problem-solving, current evaluation methodologies fail to adequately assess their capabilities: existing benchmarks either rely on closed-ended questions prone to saturation and memorization, or subjective comparisons that lack consistency and rigor. In this work, we introduce **HeuriGym**, an agentic framework designed for evaluating heuristic algorithms generated by LLMs for combinatorial optimization problems, characterized by clearly defined objectives and expansive solution spaces. HeuriGym empowers LLMs to propose heuristics, receive evaluative feedback via code execution, and iteratively refine their solutions. We evaluate nine state-of-the-art models on nine problems across domains such as computer systems, logistics, and biology, exposing persistent limitations in tool use, planning, and adaptive reasoning. To quantify performance, we propose the Quality-Yield Index (QYI), a metric that captures both solution pass rate and quality. Even top models like `GPT-o4-mini-high` and `Gemini-2.5-Pro` attain QYI scores of only 0.6, well below the expert baseline of 1. Our open-source benchmark aims to guide the development of LLMs toward more effective and realistic problem-solving in scientific and engineering domains.

1 Introduction

Recent advancements in Large Language Models (LLMs) have significantly expanded their capabilities in complex reasoning and agent-based problem-solving, enabling applications ranging from automated code generation [25, 79, 177] to dynamic decision-making systems [126, 165]. Despite these breakthroughs, existing evaluation frameworks struggle to rigorously assess the full spectrum of LLMs' emergent abilities. Traditional benchmarks increasingly fail to capture the nuanced demands of real-world tasks that require iterative reasoning, creative algorithm design, and adaptive tool use. This limitation creates a critical gap in understanding whether LLMs can transcend pattern recognition and demonstrate genuine problem-solving ingenuity in real-world scenarios.

Current evaluation paradigms fall into two categories with distinct limitations. **(1) Ground-truth-based objective benchmarks** rely on closed-form questions (e.g., multiple-choice mathematics problems) that have become susceptible to rapid performance saturation. Widely used benchmarks such as AIME [102], HumanEval [25], and GPQA Diamond [115] now exhibit ceiling effects, with state-of-the-art models achieving over 80% accuracy [103, 141, 38]. Even emerging evaluations like Humanity's Last Exam (HLE) [111], initially proposed as a rigorous PhD-level test, saw performance leap from 3% to 25% within months of release [103]. These benchmarks face a dual crisis: their static question banks risk data contamination as models ingest newer training data, while their

[*]Core Contributor

closed-ended nature fails to reflect real-world problem-solving where solutions are neither unique nor predefined. **(2) Judge-preference-based subjective evaluations**, such as Chatbot Arena [27], take a different approach by assessing model quality through pairwise comparisons by humans or LLM-based proxies [174]. These benchmarks support a wide range of plausible outputs, making them better suited for open-ended tasks. However, this flexibility introduces high variance: everyday communication tasks are inherently subjective, and judgments often prioritize superficial factors like response structure or emoji usage over substantive reasoning quality [132, 169]. While recent efforts to automate evaluation with LLM-as-a-judge systems show promise, their reliability remains inconsistent across domains [71], particularly for technical tasks requiring specialized expertise.

To address these limitations, we introduce **HeuriGym**[1], a new evaluation paradigm with an agentic framework centered on combinatorial optimization problems, which naturally combine *well-defined objectives* with *large solution spaces*. Such problems are foundational across domains including computer systems [15, 97, 88], scientific reasoning [20, 21], computational biology [153, 139, 28], logistics [77, 51], and electronic design automation [19, 33]. They are well-suited for benchmarking LLMs because they resist memorization due to their computational hardness, offer clear metrics for quantitative evaluation, and reflect real-world use cases where optimal solutions are tractable only for small instances. Importantly, no single heuristic or optimization algorithm dominates across all problems or instances [155], resulting in a rich and diverse search space. Tackling these challenges requires not only algorithmic knowledge but also heuristic reasoning, tradeoff navigation, and creative problem-solving — skills that are still underexplored in current LLM evaluations. Our framework extends beyond conventional static evaluations by implementing an interactive agentic loop: LLMs generate heuristic algorithms, receive execution feedback from a code environment, and iteratively refine their solutions. This process mirrors practical engineering workflows and enables deeper evaluation of multi-step reasoning, tool use, and instruction following.

Our benchmark systematically evaluates LLMs across four dimensions: (1) *tool-augmented reasoning* through integration with external libraries, (2) *multi-step planning* in decomposing complex problems into executable sub-tasks, (3) *instruction fidelity* in adhering to problem constraints, and (4) *iterative refinement* based on runtime feedback. The framework uniquely probes practical creativity – the ability to adapt textbook algorithms or invent novel strategies for large-scale instances where exact methods like integer linear programming (ILP) may fail.

To capture both the number of feasible solutions and their quality relative to expert performance, we introduce a unified metric – the Quality-Yield Index (QYI) – which ranges from 0 (all outputs are incorrect or low-quality) to 1 (expert-level performance). Empirical results reveal substantial performance gap: across nine diverse optimization problems, even state-of-the-art LLMs such as `GPT-o4-mini-high` [103] and `Gemini-2.5-Pro` [38] achieve QYI scores around 0.6, underscoring their limited effectiveness in realistic problem-solving settings. These findings highlight the limitations of current benchmarks, which fail to capture the complex, real-world demands of computational problem-solving—where success requires integrating theoretical understanding, tool proficiency, and adaptive reasoning. The contributions of this work are threefold:

- An open-source benchmark suite of nine combinatorial optimization problems that evaluates LLMs' multi-step reasoning capabilities through realistic programming tasks.
- An end-to-end agentic framework supporting LLM solution generation, automated verification, quantitative evaluation with well-defined metrics, and iterative refinement.
- A comprehensive empirical study of cutting-edge LLMs, uncovering their current limitations and offering actionable insights for the development of next-generation models and agents.

2 Related Work

LLMs for Combinatorial Optimization. Recent LLM-based combinatorial optimization (CO) methods follow two main paradigms. The first emphasizes formalization – translating natural language into structured optimization problems. This direction was initiated by the NL4Opt Competition [114], with follow-up work improving domain-specific model training [157, 65, 78] and prompting strategies [161, 3, 62]. While effective on benchmarks, these methods struggle to scale due to their reliance on exact solvers [53]. The second paradigm focuses on heuristic discovery. FunSearch [117] and

[1] `https://github.com/cornell-zhang/heurigym`

Table 1: Comparison with other recent benchmarks.

Subjects	Benchmark	Well-Defined Objective	Large Solution Space	Agentic Setting	Evaluation Metrics
Frontier Knowledge	Humanity's Last Exam (HLE) [111]	✓	✗	✗	Accuracy
Software Engineering	HumanEval(+) [25, 86]	✓	✗	✗	`pass@k`
	BigCodeBench [177]	✓	✗	✗	`pass@k`
	LiveCodeBench [63]	✓	✗	✗	`pass@1`
	SWE-Bench [66]	✓	✗	✗	`pass@1`
	Commit0 [171]	✓	✗	✓	Pass rate
Performance Engineering	KernelBench [106]	✗	✓	✗	$\texttt{fast}_p$
Daily-Life Tasks	Chatbot Arena [27]	✗	✓	✗	ELO
	τ-Bench [164]	✓	✓	✓	`pass^k`
Combinatorial Optimization	NPHardEval [49]	✓	✗	✗	Accuracy
	GraphArena [140]	✓	✗	✗	Accuracy
	HeuriGym (This work)	✓	✓	✓	$\texttt{solve}_s\texttt{@}i$, QYI

AlphaEvolve [100] use LLMs with evolutionary search to generate novel heuristics, but require evaluating thousands of candidates. Recent approaches [166, 84, 36, 176] improve efficiency via metaheuristic templates, but still limit LLMs to filling in scoring functions rather than designing full algorithms. In contrast, HeuriGym removes reliance on templates or scaffolds. It tasks LLMs with generating complete, self-contained optimization programs, including custom data structures and end-to-end pipelines – better reflecting real-world CO challenges, where success hinges on uncovering problem-specific structure and designing bespoke algorithms [155].

Evaluation on LLMs. As shown in Table 1, existing LLM benchmarks expose key limitations. Many focus on closed-ended tasks in domains like mathematics [102], programming [25, 177, 86], and specialized knowledge [115, 111, 57], with fixed ground-truths that are prone to data contamination (see Section 1). In contrast, open-ended benchmarks such as Chatbot Arena [27] and KernelBench [106] encourage diverse outputs but often lack clear objectives, resulting in inconsistent evaluations. Benchmarks like NPHardEval [49] and GraphArena [140] assess exact solutions to small NP-hard instances, limiting real-world relevance where heuristic solutions are often preferred for scalability. Our benchmark instead accepts any *feasible* solution that satisfies constraints, enabling broader evaluation of algorithmic reasoning. It tasks LLMs with synthesizing executable code, using external libraries, and refining solutions through execution feedback, mimicking realistic workflows. We also propose new evaluation metrics to quantify multi-round reasoning, as detailed in Section 3.2.

3 HeuriGym: An Agentic Framework for Heuristic Generation

In this section, we introduce our agentic framework for evaluating LLM reasoning via iterative heuristic generation, along with benchmark metrics for quantitative assessment.

3.1 Overview

As illustrated in Fig. 1, our framework begins by presenting a formal problem description to the LLM, which is then prompted to generate a complete heuristic algorithm. The generated program conforms to a standardized function signature and is subsequently compiled (for C++) or interpreted (for Python). Upon execution, the solution is verified for yield and evaluated for performance. Crucially, the framework incorporates a feedback loop: execution logs, verification outcomes, and evaluation costs from a small demonstration set are appended back to the prompt, enabling iterative refinement of the LLM-generated solution.

3.1.1 Problem Description

As shown on the left of Fig. 1, we use operator scheduling [33, 87], a classic optimization problem in electronic design automation, as an example. Each benchmark task is accompanied by a structured problem description with three main parts: **(1) Background**: Introduces the optimization context and key terminology to help the LLM understand the problem setting. **(2) Formalization**: Defines the optimization objective and constraints using mathematical notation (e.g., minimizing latency under

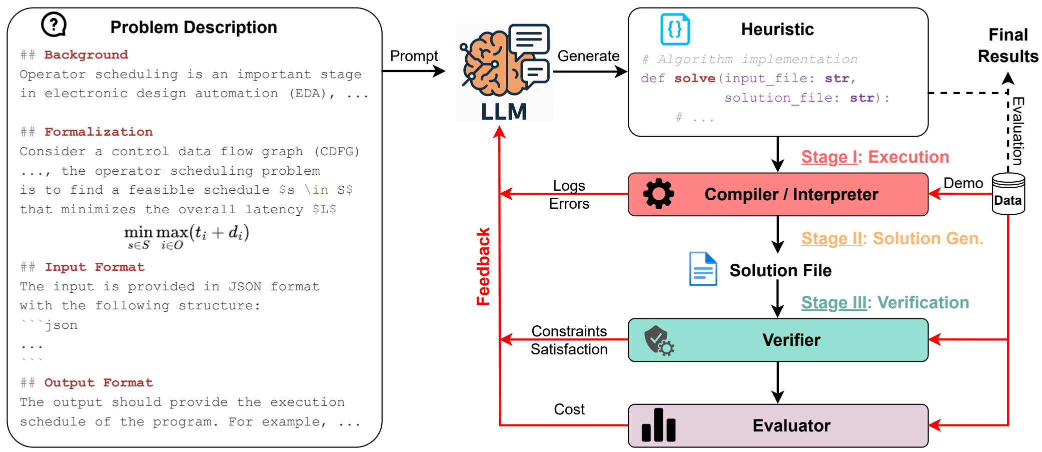

Figure 1: Overview of the HeuriGym agentic framework for heuristic program generation, execution, and verification. We use operator scheduling [33] as an example for the problem description.

hardware resource constraints), guiding the LLM toward objective-oriented algorithm design. **(3) Input/Output Format**: Specifies the structure of input and output files, providing clear expectations for parsing and execution. More detailed information on the problem set can be found in Section 4.

3.1.2 Prompt Design

Effective prompt engineering is crucial for leveraging LLMs' capabilities [152, 122]. We construct both system- and user-level prompts, tailored to each problem instance. A complete prompt example is provided in Appendix A.

System prompt. The system prompt includes machine configuration details (e.g., CPU cores, memory limits), available libraries with version numbers, and task-specific constraints such as execution timeouts. This environment specification instructs the LLM to avoid relying on unrealistic assumptions or producing inefficient solutions that violate runtime limits.

User prompt. In the initial iteration, the user prompt includes the problem description and a code skeleton with a predefined function signature. As shown in Fig. 1, the LLM is only provided the interface – function name, input path, and output path – without hints on data structures or algorithmic approache, contrasting with prior work [117, 84, 166] that often handcrafts partial implementations or restricts the design space. Here, LLMs must reason about the problem holistically: parsing inputs, constructing internal representations, and designing and implementing heuristics from scratch.

3.1.3 Feedback Loop

To emulate a few-shot in-context learning setup [42, 85, 156], we partition the dataset into a small *demonstration set* (around five instances) and a larger *evaluation set*. Demonstration data is used during the refinement loop to provide timely, example-based feedback to the LLM; the evaluation set is withheld until the model stabilizes its performance.

Each problem includes a domain-specific verifier and evaluator. The verifier ensures constraint satisfaction (e.g., dependency preservation in operator scheduling), while the evaluator calculates the cost based on the given problem objective. If the verifier fails, diagnostic messages are recorded.

After each iteration, we log the LLM-generated solution, execution trace, verification result, and evaluation score. These logs are appended to the prompt with the demonstration data in the next iteration, enabling the LLM to learn from past attempts and incrementally improve its output.

3.2 Metric Design

Traditional LLM benchmarks predominantly rely on the `pass@`k metric [25, 177, 66], which measures the probability of generating a ground-truth solution within the top-k samples. While `pass@`k is effective for single-turn tasks with deterministic ground truths, it falls short in capturing the iterative reasoning and problem-solving abilities required in our multi-round agentic setting. Specifically, it does not reflect whether the LLM can understand problem constraints, debug based on feedback, or iteratively refine its solutions over multiple attempts.

To better evaluate LLMs in this complex setting, we introduce a new metric, denoted as $\texttt{solve}_s\texttt{@}i$, which tracks the LLM's ability to solve constrained problems within i iterations:

$$\texttt{solve}_s\texttt{@}i := \frac{1}{N}\sum_{n=1}^{N} \mathbb{1}(\text{pass stage } s \text{ in the } \textit{first } i\text{-th iteration}) ,$$

where N is the total number of test instances, and $s \in \{\text{I, II, III}\}$ indicates the specific stage of the pipeline that the solution must pass. Each stage reflects a key milestone in agentic reasoning:

- **Stage I: Execution**. The generated program must compile or interpret correctly with all necessary libraries included, and successfully perform basic I/O operations (e.g., reading and writing files).
- **Stage II: Solution Generation**. The program must produce a non-empty output within the predefined timeout and adhere to the expected output format.
- **Stage III: Verification**. The solution must satisfy all problem-specific constraints, as checked by a problem-specific verifier.

However, $\texttt{solve}_s\texttt{@}i$ only indicates whether a *feasible* solution is eventually produced through the iterative process – it does not account for solution quality. To address this, we additionally define separate metrics for quality and yield as follows:

$$\texttt{Quality} = \frac{1}{\hat{N}}\sum_{n=1}^{\hat{N}} \min\left(1, \frac{c_n^\star}{c_n}\right) \qquad \texttt{Yield} = \frac{\hat{N}}{N} ,$$

where c_n and $c_n^\star$ represent the cost of the LLM-generated and expert-provided solutions, respectively, and $\hat{N}$ is the number of instances that pass verification (Stage III) in the *current* iteration. In this paper, we adopt the capped version of quality, which checks whether the LLM matches expert performance (up to a maximum of 1), though an uncapped version can also be used to measure cases where the LLM outperforms the expert. We define a unified metric, the *Quality-Yield Index (QYI)*, as the harmonic mean of quality and yield. This formulation, analogous to the F-score [144], penalizes imbalanced values more strongly than the arithmetic mean:

$$\texttt{QYI} = \frac{2 \cdot \texttt{Quality} \cdot \texttt{Yield}}{\texttt{Quality} + \texttt{Yield}} .$$

QYI captures both success rate and the relative quality of solutions, enabling holistic evaluation of an LLM's agentic reasoning capabilities, including its capacity for long-horizon planning and iterative refinement. Additionally, we can define a weighted QYI by averaging QYI scores across different problems, weighted by the number of instances in each, as an overall performance metric.

4 Benchmark Construction

This section outlines the construction of our combinatorial optimization benchmark, detailing the principles behind problem selection and providing an overview of the resulting problem set.

4.1 Problem Selection Criteria

Our primary goal is to evaluate an LLM's capacity for reasoning rather than its ability to regurgitate well-known algorithms. To this end, we intentionally exclude ubiquitous problems such as the Traveling Salesman Problem [116] and canonical satisfiability (SAT) formulations [125] – problems that are so widely studied and frequently included in public datasets that they are likely memorized during pretraining. Instead, we focus on problems that meet the following criteria:

Limited exposure in the literature. For each candidate problem, we perform a Google Scholar search and retain it only if the most-cited paper has fewer than 1,000 citations (as of April 2025). This

Table 2: Existing combinatorial optimization problems in our HeuriGym benchmark.

Domain	Problem	References	Difficulty
Electronic Design Automation (EDA)	Operator scheduling	[33, 129, 87]	★
	Technology mapping	[19, 95]	★★
	Global routing	[80, 81]	★★★
Compilers	E-graph extraction	[15, 50, 154]	★
	Intra-operator parallelism	[97, 175, 44]	★★
Computational Biology	Protein sequence design	[139, 56, 70]	★
	Mendelian error detection	[153, 123, 101]	★★
Logistics	Airline crew pairing	[51, 2, 90]	★★
	Pickup and delivery w/ time windows	[77, 46]	★★★

empirical threshold ensures that the problem is well-defined and supported by peer-reviewed work, yet not so well-known that an LLM could solve it through rote memorization or pattern matching.

Clear natural-language specification with well-defined objectives. Each problem must be clearly expressible using plain language without the need for visual aids. We encode mathematical objectives in LaTeX to eliminate ambiguity, ensuring the LLM receives well-specified instructions.

Large solution spaces. We focus on problems that admit vast solution spaces with many feasible outputs, encouraging creative exploration and reasoning rather than narrow pattern recognition [60].

Scalable data instances. Each problem includes two disjoint sets of instances: a small-scale demonstration set and a large-scale evaluation set, differing by at least an order of magnitude. The demonstration set supports few-shot prompting and iterative refinement, while the evaluation set is reserved for final performance testing, as discussed in Section 3.1.3.

Reproducible expert baselines. Reference implementations are bundled in the benchmark repository to ensure fair comparison across future studies. Where possible, we include both exact solvers (e.g., ILP) and high-quality heuristics to illuminate the performance gap.

We prioritize domains with real-world impact, where even small gains yield significant societal or industrial benefits. Many selected problems remain open, with heuristics far from theoretical bounds – offering a compelling testbed for LLMs.

4.2 Dataset Statistics

The initial release of the HeuriGym benchmark includes nine distinct optimization problems spanning four scientific and engineering domains, as summarized in Table 2. For each problem, we provide around five demonstration instances and 20 large-scale evaluation instances, totaling 218 data instances. All datasets are derived from realistic sources and real-world applications, enhancing the benchmark's practical relevance. In addition, we reserve hundreds of instances as private test sets for future release and evaluation.

A detailed description of each problem is provided in Appendix C. Notably, most problems in the benchmark are NP-hard and feature complex constraints, resulting in a compact yet highly challenging problem suite. Despite its modest size, the benchmark still presents substantial difficulty for current state-of-the-art LLMs, as shown in Section 5.

To ensure clarity and correctness, we adopt a human-in-the-loop process for problem specification. After drafting the initial natural-language description, an annotator prompts a weaker LLM [83] to identify any unclear or ambiguous statements. Discrepancies are iteratively resolved until the description is unambiguous and fully aligned with the intended semantics. The full prompt template used for refining problem descriptions is provided in Appendix A.

Each problem includes a task-specific verifier and evaluator to assess solution pass rate and quality. A separate reviewer ensures the expert solver reproduces published results and passes both checks.

Looking forward, we plan to extend HeuriGym along two axes: (1) *breadth*, by incorporating additional combinatorial optimization problems from underexplored scientific domains; and (2) *depth*, by scaling existing problems to larger instance sizes and tighter constraint settings. Community contributions are welcome, provided new problems satisfy the selection criteria articulated above.

Table 3: Overall $\texttt{solve}_s\texttt{@}i$ metric of models on the whole HeuriGym benchmark.

Model	$\texttt{solve}_{\text{III}}$ @10	$\texttt{solve}_{\text{III}}$ @5	$\texttt{solve}_{\text{III}}$ @1	$\texttt{solve}_{\text{II}}$ @10	$\texttt{solve}_{\text{II}}$ @5	$\texttt{solve}_{\text{II}}$ @1	$\texttt{solve}_{\text{I}}$ @10	$\texttt{solve}_{\text{I}}$ @5	$\texttt{solve}_{\text{I}}$ @1
`DeepSeek-V3`	46.8%	42.7%	14.2%	87.6%	83.0%	66.1%	**100.0%**	**100.0%**	90.8%
`DeepSeek-R1`	73.4%	**72.9%**	44.0%	88.1%	88.1%	60.6%	**100.0%**	**100.0%**	71.6%
`Gemini-2.5-Flash`	67.4%	58.3%	25.2%	83.9%	79.4%	56.4%	**100.0%**	**100.0%**	72.9%
`Gemini-2.5-Pro`	65.1%	64.2%	20.2%	89.4%	89.0%	42.7%	**100.0%**	**100.0%**	51.4%
`LLaMA-4-Maverick`	35.8%	33.5%	6.0%	84.9%	74.3%	8.3%	85.3%	85.3%	13.3%
`LLaMA-3.3-70B`	33.9%	33.9%	20.6%	78.4%	78.4%	40.4%	99.5%	99.5%	61.9%
`Qwen3-235B`	45.9%	45.4%	38.5%	86.2%	83.0%	56.0%	**100.0%**	**100.0%**	70.6%
`Claude-3.7-Sonnet`	60.1%	58.7%	9.2%	97.7%	97.7%	41.3%	**100.0%**	**100.0%**	60.1%
`GPT-o4-mini`	**74.8%**	69.7%	**53.2%**	**100.0%**	**100.0%**	**93.1%**	**100.0%**	**100.0%**	**100.0%**

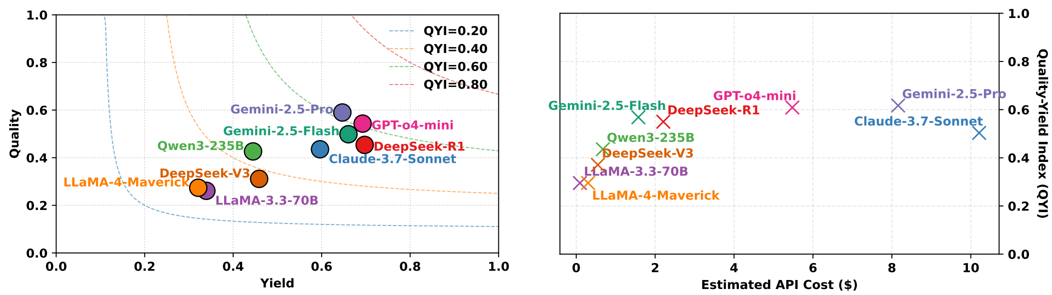

Figure 2: Quality-Yield Index and estimated API cost of different models.

5 Evaluation

To evaluate the reasoning capabilities of LLMs on combinatorial optimization problems, we benchmark nine prominent models released in late 2024 and early 2025. These include OpenAI `GPT-o4-mini-high` [103], Anthropic `Claude-3.7-Sonnet` [9], `DeepSeek-V3` [83] and `DeepSeek-R1` [52], Google `Gemini-2.5-Flash` and `Pro` [38], Meta `LLaMA-3.3` [91] and `LLaMA-4-Maverick` [92], and Alibaba `Qwen3-235B` [141]. These models represent the current state-of-the-art in general-purpose LLMs and rank among the top entries on OpenRouter [104] and Chatbot Arena leaderboards [27]. We exclude smaller models due to the complexity of the benchmark tasks. Detailed model specifications are provided in Appendix B.

All evaluations are conducted via official APIs to ensure reproducibility. We adopt the agentic workflow in Fig. 1, constraining each model to generate Python programs that solve the given problems under fixed resource limits: a maximum of 8 CPU cores and problem-specific timeouts. We also allow the models to access external libraries like `numpy` and `pandas` for simple tool use. We then assess model performance through both quantitative metrics and qualitative case analysis. Full details of the experimental settings and results can be found in Appendix D.

5.1 Overall Performance

For the overall evaluation, we fix the generation temperature at 0, following standard practice in recent LLM benchmarks [106, 164, 111]. This ensures deterministic outputs and eliminates randomness across runs. Notably, OpenAI's o-series models only support a fixed temperature of 1.0 [103]. We measure the multi-round performance using the $\texttt{solve}_s\texttt{@}i$ metric, where i indicates the number of iterations allowed (we use $i = 1, 5$, and 10).

As shown in Table 3, most LLMs fail to solve a large fraction of test cases within a single attempt, as reflected in the $\texttt{solve}_{\text{III}}\texttt{@1}$ score. Increasing the number of iterations generally improves performance across all models. For instance, the $\texttt{solve}_{\text{III}}$ success rate rises from 53.2% to 74.8% for `GPT-o4-mini` as i increases, underscoring the importance of iterative refinement in improving LLM-generated solutions. Among all models, `GPT-o4-mini` and `DeepSeek-R1` demonstrate high success rates across multiple iterations, highlighting their stronger program repair capabilities.

To assess solution quality, we compare the final LLM-generated programs to expert-designed solutions using the weighted QYI metric defined in Section 3.2. As illustrated in Fig. 2, a substantial

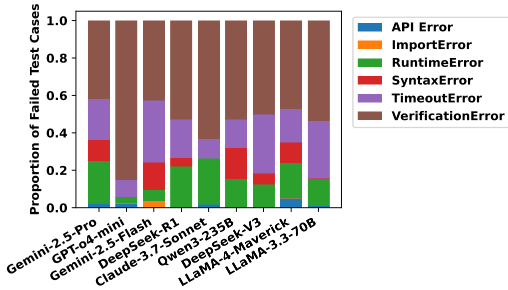

Figure 3: Error classifications.

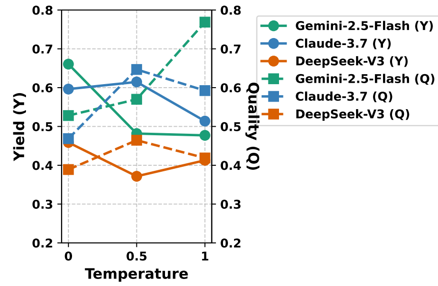

Figure 4: Quality-Yield tradeoff.

performance gap remains: even the best-performing model, `Gemini-2.5-Pro`, achieves a QYI of only 0.62, indicating that its solutions are, on average, just 60% as effective as expert-crafted ones. Several models, such as `LLaMA-3.3` and `LLaMA-4`, produce results with QYI scores below 30%, highlighting their limited effectiveness on these tasks. We also estimate the API cost for each model and find that `Gemini-2.5-Flash` offers the best cost-efficiency relative to its achieved QYI.

To identify common failure modes, we analyze and categorize the most common error types produced by the evaluated models, as shown in Fig. 3. These include: (1) Hallucinated APIs: using nonexistent or outdated library calls. (2) Incorrect algorithmic logic: flawed implementation even when the general approach is reasonable. (3) Constraint misunderstanding: ignoring or misinterpreting problem constraints. (4) Timeouts: no output or the execution time exceeds the given constraints. Additional error cases and examples are listed in Appendix D.

5.2 Ablation Study

To assess the robustness and sensitivity of LLM performance under different settings, we conduct a set of ablation experiments with full details in Appendix D.

Temperature. We evaluate three representative models across the QYI spectrum using decoding temperatures $T \in \{0.0, 0.5, 1.0\}$. As shown in Fig. 4, higher T increases diversity and quality but lowers yield due to more invalid outputs. Greedy decoding ($T = 0$) has maximum yield with suboptimal quality, while stochastic sampling ($T = 1$) achieves better quality at the cost of solving fewer problems. Our benchmark reveals a fundamental trade-off between quality and yield that future LLMs must address.

Table 4: Ablation study on the pickup and delivery with time windows problem [77, 46].

# of Demos / # of Feedback Rounds	5/10	3/10	0/10	5/5	5/1
QYI Score	0.4196	0.2829	0.2351	0.3330	0.2350

Few-shot demonstrations. We assess the impact of in-context examples by comparing zero-shot, half-shot, and full-shot prompts. Due to budget constraints, these experiments are conducted on a few representative models. Specifically, we evaluate `Gemini-2.5-Pro` on the pickup and delivery problem – one of the most challenging tasks in our benchmark. As shown in Table 4, providing more informative demonstrations significantly boosts the overall performance, especially for tasks involving unfamiliar domains or requiring long-horizon reasoning.

Feedback rounds. To evaluate the role of iterative refinement, we vary the number of feedback rounds given to LLMs (1, 5, and 10), keeping the temperature fixed at 0. The results in Table 4 show that later iterations frequently fix logic errors or constraint violations from earlier attempts, underscoring the value of multi-round reasoning. We provide further analysis in Section 5.3.

5.3 Case Study

We present a case study on technology mapping [95] to highlight both the promise and current limitations of LLMs. The goal is to cover a logic network with K-input subgraphs – corresponding to lookup tables (LUTs) – that minimize the total number of LUTs. We fix $K = 6$ in our setting.

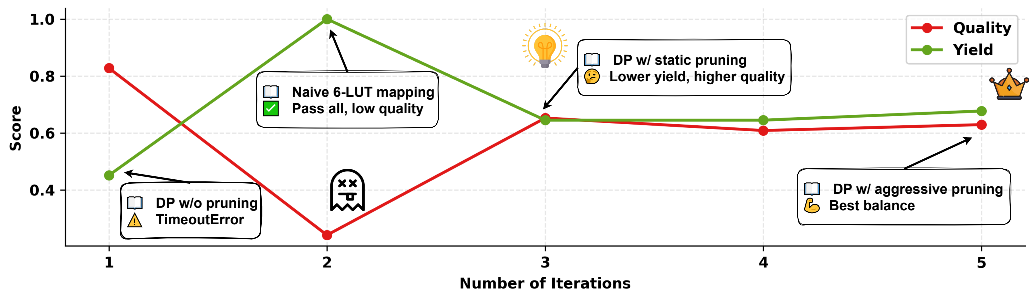

Figure 5: One iterative example of `GPT-o4-mini` on the technology mapping problem.

As an expert baseline, we use ABC [14], a state-of-the-art logic synthesis tool that leverages optimized cut enumeration and dynamic programming (DP)-based covering. We find that top-performing LLMs, such as `GPT-o4-mini` and `Gemini-2.5-Pro`, can mimic similar heuristic strategies and iteratively refine them through feedback. As illustrated in Fig. 5, `GPT-o4-mini` explores a range of approaches over multiple iterations, evolving from naive mappings to increasingly sophisticated DP-based heuristics with pruning. By the fifth iteration, it converges on a strategy that effectively balances yield and solution quality, which achieves the highest QYI score.

Nonetheless, a substantial gap remains between LLMs and expert tools, due to the latter's extensive use of domain-specific optimizations and efficient implementations. This suggests that while LLMs can learn and refine heuristic algorithms, they are not yet capable of generating solutions with expert-level performance in real-world complex optimization tasks.

6 Discussion and Limitation

While our benchmark and framework offer a promising foundation for evaluating LLMs on combinatorial optimization problems, several limitations remain that suggest directions for future work.

First, all experiments are conducted in Python, which, while accessible, introduces execution overhead at scale. Although we include preliminary results with C++ in Appendix D, integrating C++ remains challenging due to dependencies on domain-specific libraries and the difficulty LLMs face in generating efficient, functionally correct, and parallel C++ code.

Second, the iterative self-refinement process in our agentic workflow can be interpreted as a form of test-time scaling (TTS), analogous to compute-optimal scaling strategies [134]. This perspective creates opportunities to incorporate techniques such as Best-of-N sampling [138], beam search [158], and evolutionary algorithms [100, 166], especially with increased iteration budgets. Furthermore, with a robust verifier in place, our framework provides a natural platform to investigate self-verification capabilities [72, 168, 173], a promising avenue toward greater LLM autonomy.

Third, our evaluation pipeline currently relies on proxy metrics that are formally defined and computationally efficient. While these metrics are useful for initial benchmarking, they often fall short in reflecting real-world performance – particularly in (1) scientific domains, where solution quality must ultimately be validated through physical experiments, and (2) engineering domains like EDA, where quality must be confirmed through time-consuming backend synthesis. Bridging the gap between proxy evaluation and real-world deployment – while managing the latency introduced by longer feedback loops – remains a key challenge and an important direction for future work.

Currently HeuriGym includes only nine problems. Although these have been carefully curated to test reasoning and generalization, they may eventually become saturated as LLM capabilities improve. To maintain long-term relevance, we envision the benchmark as an evolving resource and actively encourage community contributions. Future iterations will expand both the breadth and depth of the benchmark by introducing new problems from underrepresented scientific domains and scaling existing ones to more complex regimes.

By formalizing combinatorial problems with standardized formats with reproducible agentic workflows, we believe HeuriGym can serve as a shared testbed and foster interdisciplinary collaboration.

References

[1] Luca Aceto, Jens A Hansen, Anna Ingólfsdóttir, Jacob Johnsen, and John Knudsen. The complexity of checking consistency of pedigree information and related problems. *Journal of Computer Science and Technology*, 19:42–59, 2004.

[2] Divyam Aggarwal, Dhish Kumar Saxena, Thomas Bäck, and Michael Emmerich. Real-world airline crew pairing optimization: Customized genetic algorithm versus column generation method. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 518–531. Springer, 2023.

[3] Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. Optimus: scalable optimization modeling with (mi)lp solvers and large language models. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

[4] Christoph Albrecht. Global routing by new approximation algorithms for multicommodity flow. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(5):622–632, 2001.

[5] Benjamin D Allen and Stephen L Mayo. An efficient algorithm for multistate protein design based on faster. *Journal of computational chemistry*, 31(5):904–916, 2010.

[6] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. The epfl combinational benchmark suite. In *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, 2015.

[7] Xavier I Ambroggio and Brian Kuhlman. Computational design of a single amino acid sequence that can switch between two distinct protein folds. *Journal of the American Chemical Society*, 128(4):1154–1161, 2006.

[8] Ranga Anbil, Rajan Tanga, and Ellis L. Johnson. A global approach to crew-pairing optimization. *IBM Systems Journal*, 31(1):71–78, 1992.

[9] Anthropic. Claude 3.7 sonnet and claude code, 2025. `https://www.anthropic.com/news/claude-3-7-sonnet`.

[10] Roberto Baldacci, Enrico Bartolini, and Aristide Mingozzi. An exact algorithm for the pickup and delivery problem with time windows. *Operations research*, 59(2):414–426, 2011.

[11] Jayanth R Banavar, Marek Cieplak, Amos Maritan, Gautham Nadig, Flavio Seno, and Saraswathi Vishveshwara. Structure-based design of model proteins. *Proteins: Structure, Function, and Bioinformatics*, 31(1):10–20, 1998.

[12] Laleh Behjat, Anthony Vannelli, and William Rosehart. Integer linear programming models for global routing. *INFORMS Journal on Computing*, 18(2):137–150, 2006.

[13] Russell Bent and Pascal Van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, 33(4):875–893, 2006.

[14] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification, 2005. `http://www.eecs.berkeley.edu/~alanmi/abc/`.

[15] Yaohui Cai, Kaixin Yang, Chenhui Deng, Cunxi Yu, and Zhiru Zhang. Smoothe: Differentiable e-graph extraction. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 1*, 2025.

[16] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H Anderson, Stephen Brown, and Tomasz Czajkowski. Legup: high-level synthesis for fpga-based processor/accelerator systems. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, pages 33–36, 2011.

[17] RC Carden, Jianmin Li, and Chung-Kuan Cheng. A global router with a theoretical bound on the optimal solution. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(2):208–216, 1996.

[18] Chong-Yun Chao and Earl Glen Whitehead. On chromatic equivalence of graphs. *Theory and Applications of Graphs*, pages 121–131, 1978.

[19] Deming Chen and Jason Cong. Daomap: A depth-optimal area optimization mapping algorithm for fpga designs. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, pages 752–759. IEEE, 2004.

[20] Di Chen, Yiwei Bai, Sebastian Ament, Wenting Zhao, Dan Guevarra, Lan Zhou, Bart Selman, R Bruce van Dover, John M Gregoire, and Carla P Gomes. Automating crystal-structure phase mapping by combining deep learning with constraint reasoning. *Nature Machine Intelligence*, 3(9):812–822, 2021.

[21] Di Chen, Yexiang Xue, Shuo Chen, Daniel Fink, and Carla Gomes. Deep multi-species embedding. *arXiv preprint arXiv:1609.09353*, 2016.

[22] Hongzheng Chen and Minghua Shen. A deep-reinforcement-learning-based scheduler for fpga hls. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.

[23] Hongzheng Chen, Cody Hao Yu, Shuai Zheng, Zhen Zhang, Zhiru Zhang, and Yida Wang. Slapo: A schedule language for progressive optimization of large deep learning model training. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 1095–1111, 2024.

[24] Hongzheng Chen, Niansong Zhang, Shaojie Xiang, Zhichen Zeng, Mengjia Dai, and Zhiru Zhang. Allo: A programming model for composable accelerator design. *Proceedings of the ACM on Programming Languages*, 8(PLDI):593–620, 2024.

[25] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[26] Jianyi Cheng, Samuel Coward, Lorenzo Chelini, Rafael Barbalho, and Theo Drane. Seer: Super-optimization explorer for high-level synthesis using e-graph rewriting. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 1029–1044, 2024.

[27] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios N. Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael I. Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: an open platform for evaluating llms by human preference. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

[28] Michael Codish, Michael Frank, and Vitaly Lagoon. The dna word design problem: A new constraint model and new results. In *IJCAI*, pages 585–591, 2017.

[29] China Graduate Mathematical Modeling Competition. Problem f, 2021.

[30] Jason Cong and Yuzheng Ding. Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(1):1–12, 1994.

[31] Jason Cong, Bin Liu, Stephen Neuendorffer, Juanjo Noguera, Kees Vissers, and Zhiru Zhang. High-level synthesis for fpgas: From prototyping to deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):473–491, 2011.

[32] Jason Cong and Patrick Madden. Performance-driven global routing for standard cell design. In *Proceedings of the 1998 International Symposium on Physical Design*, pages 73–78, 1998.

[33] Jason Cong and Zhiru Zhang. An efficient and versatile scheduling algorithm based on sdc formulation. In *Proceedings of the 43rd annual Design Automation Conference (DAC)*, pages 433–438, 2006.

[34] Timothy Curtois, Dario Landa-Silva, Yi Qu, and Wasakorn Laesanklang. Large neighbourhood search with adaptive guided ejection search for the pickup and delivery problem with time windows. *EURO Journal on Transportation and Logistics*, 7(2):151–192, 2018.

[35] Steve Dai, Gai Liu, and Zhiru Zhang. A scalable approach to exact resource-constrained scheduling based on a joint sdc and sat formulation. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 137–146, 2018.

[36] Pham Vu Tuan Dat, Long Doan, and Huynh Thi Thanh Binh. Hsevo: Elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using llms. In *The 39th Annual AAAI Conference on Artificial Intelligence*, 2025. `https://github.com/datphamvn/HSEvo`.

[37] Protein Database. Rcsb protein data bank (rcsb pdb), 2024.

[38] Google DeepMind. Gemini 2.5: Our most intelligent ai model, 2025. `https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025`.

[39] Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2006.

[40] JM Deutsch and Tanya Kurosky. New algorithm for protein design. *Physical review letters*, 76(2):323, 1996.

[41] Ken A Dill, Sarina Bromberg, Kaizhi Yue, Hue Sun Chan, Klaus M Ftebig, David P Yee, and Paul D Thomas. Principles of protein folding—a perspective from simple exact models. *Protein science*, 4(4):561–602, 1995.

[42] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. A survey on in-context learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024.

[43] K Eric Drexler. Molecular engineering: An approach to the development of general capabilities for molecular manipulation. *Proceedings of the National Academy of Sciences*, 78(9):5275–5278, 1981.

[44] Jiangsu Du, Jinhui Wei, Jiazhi Jiang, Shenggan Cheng, Dan Huang, Zhiguang Chen, and Yutong Lu. Liger: Interleaving intra-and inter-operator parallelism for distributed large model inference. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, pages 42–54, 2024.

[45] Yuanqi Du, Arian R Jamasb, Jeff Guo, Tianfan Fu, Charles Harris, Yingheng Wang, Chenru Duan, Pietro Liò, Philippe Schwaller, and Tom L Blundell. Machine learning-aided generative molecular design. *Nature Machine Intelligence*, 6(6):589–604, 2024.

[46] Yvan Dumas, Jacques Desrosiers, and Francois Soumis. The pickup and delivery problem with time windows. *European journal of operational research*, 54(1):7–22, 1991.

[47] Yvan Dumas, Jacques Desrosiers, and Francois Soumis. The pickup and delivery problem with time windows. *European journal of operational research*, 54(1):7–22, 1991.

[48] Issmail Elhallaoui, Daniel Villeneuve, François Soumis, and Guy Desaulniers. Dynamic aggregation of set-partitioning constraints in column generation. *Operations Research*, 53(4):632–645, 2005.

[49] Lizhou Fan, Wenyue Hua, Lingyao Li, Haoyang Ling, and Yongfeng Zhang. NPHardEval: Dynamic benchmark on reasoning ability of large language models via complexity classes. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024.

[50] Amir Kafshdar Goharshady, Chun Kit Lam, and Lionel Parreaux. Fast and optimal extraction for sparse equality graphs. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA2):2551–2577, 2024.

[51] Glenn W Graves, Richard D McBride, Ira Gershkoff, Diane Anderson, and Deepa Mahidhara. Flight crew scheduling. *Management science*, 39(6):736–745, 1993.

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

[53] Gurobi. Gurobi optimizer, 2025. `https://www.gurobi.com/solutions/gurobi-optimizer/`.

[54] Mark A Hallen and Bruce R Donald. Comets (constrained optimization of multistate energies by tree search): A provable and efficient protein design algorithm to optimize binding affinity and specificity with respect to sequence. *Journal of Computational Biology*, 23(5):311–321, 2016.

[55] Mark C Hansen, Hakan Yalcin, and John P Hayes. Unveiling the iscas-85 benchmarks: A case study in reverse engineering. *IEEE Design & Test of Computers*, 1999.

[56] William E Hart. On the computational complexity of sequence design problems. In *Proceedings of the first annual international conference on Computational molecular biology*, pages 128–136, 1997.

[57] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[58] Sin C Ho, Wai Yuen Szeto, Yong-Hong Kuo, Janny MY Leung, Matthew Petering, and Terence WH Tou. A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, 111:395–421, 2018.

[59] Jiang Hu and Sachin S Sapatnekar. A survey on multi-net global routing for integrated circuits. *Integration*, 31(1):1–49, 2001.

[60] Edward Hughes, Michael D Dennis, Jack Parker-Holder, Feryal Behbahani, Aditi Mavalankar, Yuge Shi, Tom Schaul, and Tim Rocktäschel. Position: Open-endedness is essential for artificial superhuman intelligence. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

[61] C-T Hwang, J-H Lee, and Y-C Hsu. A formal approach to the scheduling problem in high level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(4):464–475, 1991.

[62] Zangir Iklassov, Yali Du, Farkhad Akimov, and Martin Takáč. Self-guiding exploration for combinatorial problems. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[63] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[64] Jeppesen. Jeppesen crew pairing solution. `https://ww2.jeppesen.com/airline-crew-optimization-solutions/airline-crew-pairing/`, 2021.

[65] Caigao JIANG, Xiang Shu, Hong Qian, Xingyu Lu, JUN ZHOU, Aimin Zhou, and Yang Yu. LLMOPT: Learning to define and solve general optimization problems from scratch. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[66] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024.

[67] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

[68] Satwik Kamtekar, Jarad M Schiffer, Huayu Xiong, Jennifer M Babik, and Michael H Hecht. Protein design by binary patterning of polar and nonpolar amino acids. *Science*, 262(5140):1680–1685, 1993.

[69] Atoosa Kasirzadeh, Mohammed Saddoune, and François Soumis. Airline crew scheduling: models, algorithms, and data sets. *EURO Journal on Transportation and Logistics*, 6(2):111–137, 2017.

[70] Jon M Kleinberg. Efficient algorithms for protein sequence design and the analysis of certain evolutionary fitness landscapes. In *Proceedings of the third annual international conference on Computational molecular biology*, pages 226–237, 1999.

[71] Michael Krumdick, Charles Lovering, Varshini Reddy, Seth Ebner, and Chris Tanner. No free labels: Limitations of llm-as-a-judge without human grounding. *arXiv preprint arXiv:2503.05061*, 2025.

[72] Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024.

[73] Avery Laird, Bangtian Liu, NIKOLAJ BJØRNER, and Maryam Mehri Dehnavi. Speq: Translation of sparse codes using equivalences. *ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*, 2024.

[74] Kit Fun Lau and Ken A Dill. Theory for protein mutability and biogenesis. *Proceedings of the National Academy of Sciences*, 87(2):638–642, 1990.

[75] Chin Yang Lee. An algorithm for path connections and its applications. *IRE transactions on electronic computers*, EC-10(3):346–365, 2009.

[76] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[77] Haibing Li and Andrew Lim. A metaheuristic for the pickup and delivery problem with time windows. In *Proceedings 13th IEEE international conference on tools with artificial intelligence*, pages 160–167. IEEE, 2001.

[78] Sirui Li, Janardhan Kulkarni, Ishai Menache, Cathy Wu, and Beibin Li. Towards foundation models for mixed integer linear programming. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[79] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

[80] Rongjian Liang, Anthony Agnesina, Wen-Hao Liu, and Haoxing Ren. Gpu/ml-enhanced large scale global routing contest. In *Proceedings of the 2024 International Symposium on Physical Design (ISPD)*, 2024.

[81] Haiguang Liao, Wentai Zhang, Xuliang Dong, Barnabas Poczos, Kenji Shimada, and Levent Burak Kara. A deep reinforcement learning approach for global routing. *Journal of Mechanical Design*, 142(6):061701, 2020.

[82] Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, et al. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, 2023.

[83] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[84] Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

[85] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for GPT-3? In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, 2022.

[86] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[87] Mingju Liu, Yingjie Li, Jiaqi Yin, Zhiru Zhang, and Cunxi Yu. Differentiable combinatorial scheduling at scale. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

[88] Tianfeng Liu, Yangrui Chen, Dan Li, Chuan Wu, Yibo Zhu, Jun He, Yanghua Peng, Hongzheng Chen, Hongzhi Chen, and Chuanxiong Guo. BGL:GPU-efficient GNN training by optimizing graph data I/O and preprocessing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 103–118, 2023.

[89] Panta Lučić and Dušan Teodorović. Metaheuristics approach to the aircrew rostering problem. *Annals of Operations Research*, 155:311–338, 2007.

[90] Xiaodong Luo, Yogesh Dashora, and Tina Shaw. Airline crew augmentation: decades of improvements from sabre. *Interfaces*, 45(5):409–424, 2015.

[91] Meta. The llama 3 herd of models, 2024. `https://ai.meta.com/research/publications/the-llama-3-herd-of-models/`.

[92] Meta. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation, 2025. `https://ai.meta.com/blog/llama-4-multimodal-intelligence/`.

[93] Cristian Micheletti, Jayanth R Banavar, Amos Maritan, and Flavio Seno. Protein structures and optimal folding from a geometrical variational principle. *Physical Review Letters*, 82(16):3372, 1999.

[94] Yaosen Min, Ye Wei, Peizhuo Wang, Xiaoting Wang, Han Li, Nian Wu, Stefan Bauer, Shuxin Zheng, Yu Shi, Yingheng Wang, et al. From static to dynamic structures: Improving binding affinity prediction with a graph-based deep learning model. *arXiv e-prints*, pages arXiv–2208, 2022.

[95] Alan Mishchenko, Satrajit Chatterjee, and Robert Brayton. Improvements to technology mapping for lut-based fpgas. In *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays (FPGA)*, 2006.

[96] Michael D Moffitt. Global routing revisited. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 805–808, 2009.

[97] Michael D. Moffitt and Pratik Fegade. The asplos 2025 / eurosys 2025 contest on intra-operator parallelism for distributed deep learning. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, volume 3 of *ASPLOS 2025*, 2025.

[98] Christopher Negron and Amy E Keating. Multistate protein design using clever and classy. In *Methods in enzymology*, volume 523, pages 171–190. Elsevier, 2013.

[99] Greg Nelson and Derek C Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.

[100] Alexander Novikov, Ngân Vu, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. Technical report, Technical report, Google DeepMind, 05 2025. URL https://storage. googleapis ..., 2025.

[101] Jeffrey R O'Connell and Daniel E Weeks. An optimal algorithm for automatic genotype elimination. *The American Journal of Human Genetics*, 65(6):1733–1740, 1999.

[102] AoPS Online. American invitational mathematics examination (aime), 2025. `https://artofproblemsolving.com/wiki/index.php/American_Invitational_Mathematics_Examination`.

[103] OpenAI. Introducing openai o3 and o4-mini, 2025. `https://openai.com/index/introducing-o3-and-o4-mini/`.

[104] OpenRouter. Llm rankings, 2025. `https://openrouter.ai/rankings`.

[105] Julian Oppermann, Andreas Koch, Melanie Reuter-Oppermann, and Oliver Sinnen. Ilp-based modulo scheduling for high-level synthesis. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pages 1–10, 2016.

[106] Anne Ouyang, Simon Guo, Simran Arora, Alex L. Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. Kernelbench: Can llms write efficient gpu kernels?, 2025.

[107] Debjit Pal, Yi-Hsiang Lai, Shaojie Xiang, Niansong Zhang, Hongzheng Chen, Jeremy Casas, Pasquale Cocchini, Zhenkun Yang, Jin Yang, Louis-Noël Pouchet, et al. Accelerator design with decoupled hardware customizations: benefits and challenges. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 1351–1354, 2022.

[108] Pavel Panchekha, Alex Sanchez-Stern, James R Wilcox, and Zachary Tatlock. Automatically improving accuracy for floating point expressions. *ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*, 50(6):1–11, 2015.

[109] Alice C Parker, Jorge Pizarro, and Mitch Mlinar. Maha: A program for datapath synthesis. In *23rd ACM/IEEE Design Automation Conference*, pages 461–466. IEEE, 1986.

[110] Pierre G Paulin and John P Knight. Force-directed scheduling for the behavioral synthesis of asics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(6):661–679, 2002.

[111] Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

[112] Navin Pokala and Tracy M Handel. Energy functions for protein design: adjustment with protein–protein complex affinities, models for the unfolded state, and negative design of solubility and specificity. *Journal of molecular biology*, 347(1):203–227, 2005.

[113] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

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

[115] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

[116] Julia Robinson. *On the Hamiltonian game (a traveling salesman problem)*. Rand Corporation, 1949.

[117] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[118] Stefan Ropke and Jean-François Cordeau. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation science*, 43(3):267–286, 2009.

[119] Stefan Ropke, Jean-François Cordeau, and Gilbert Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks: An International Journal*, 49(4):258–272, 2007.

[120] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006.

[121] Sabre. Sabre crew pairing. `https://your.sabre.com/inthistogether/restart_efficient_ops`, 2020.

[122] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*, 2024.

[123] Marti Sanchez, Simon de Givry, and Thomas Schiex. Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. *Constraints*, 13(1):130–154, 2008.

[124] Carlo S Sartori and Luciana S Buriol. A study on the pickup and delivery problem with time windows: Matheuristics and new instances. *Computers & Operations Research*, 124:105065, 2020.

[125] Thomas J Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226, 1978.

[126] Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[127] Thomas Schiex. Cost function library. `https://forgemia.inra.fr/thomas.schiex/cost-function-library`, 2018.

[128] Eugene I Shakhnovich and AM Gutin. A new approach to the design of stable proteins. *Protein Engineering, Design and Selection*, 6(8):793–800, 1993.

[129] Minghua Shen, Hongzheng Chen, and Nong Xiao. Entropy-directed scheduling for fpga high-level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2588–2601, 2019.

[130] Ziji Shi, Le Jiang, Ang Wang, Jie Zhang, Xianyan Jia, Yong Li, Chencan Wu, Jialin Li, and Wei Lin. Tap: Efficient derivation of tensor parallel plans for large neural networks. In *Architecture and System Support for Transformer Models (ASSYST@ ISCA)*, 2023.

[131] Eugene Shragowitz and Lynn Keel. A multicommodity flow approach to concurrent global routing. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pages 414–419, 1987.

[132] Shivalika Singh, Yiyang Nan, Alex Wang, Daniel D'Souza, Sayash Kapoor, Ahmet Üstün, Sanmi Koyejo, Yuntian Deng, Shayne Longpre, Noah Smith, et al. The leaderboard illusion. *arXiv preprint arXiv:2504.20879*, 2025.

[133] Gus Henry Smith, Zachary D Sisco, Thanawat Techaumnuaiwit, Jingtao Xia, Vishal Canumalla, Andrew Cheung, Zachary Tatlock, Chandrakana Nandi, and Jonathan Balkind. There and back again: A netlist's tale with much egraphin'. *arXiv preprint arXiv:2404.00786*, 2024.

[134] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[135] Nadia Souai and Jacques Teghem. Genetic algorithm based approach for the integrated airline crew-pairing and rostering problem. *European Journal of Operational Research*, 199(3):674–683, 2009.

[136] JIRI Soukup. Fast maze router. In *Design Automation Conference*, pages 100–101. IEEE Computer Society, 1978.

[137] Michael B. Stepp. *Equality Saturation: Engineering Challenges and Applications*. PhD thesis, University of California San Diego, 2011.

[138] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021, 2020.

[139] Shaojian Sun, Rachel Brem, Hue Sun Chan, and Ken A Dill. Designing amino acid sequences to fold with good hydrophobic cores. *Protein Engineering, Design and Selection*, 8(12):1205–1213, 1995.

[140] Jianheng Tang, Qifan Zhang, Yuhan Li, Nuo Chen, and Jia Li. Grapharena: Evaluating and exploring large language models on graph computation. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[141] Qwen Team. Qwen3: Think deeper, act faster, 2025. `https://qwenlm.github.io/blog/qwen3/`.

[142] Samuel Thomas and James Bornholt. Automatic generation of vectorizing compilers for customizable digital signal processors. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 19–34, 2024.

[143] Ecenur Ustun, Ismail San, Jiaqi Yin, Cunxi Yu, and Zhiru Zhang. Impress: Large integer multiplication expression rewriting for fpga hls. *IEEE Symp. on Field Programmable Custom Computing Machines (FCCM)*, pages 1–10, 2022.

[144] C Van Rijsbergen. Information retrieval: theory and practice. In *Proceedings of the joint IBM/University of Newcastle upon tyne seminar on data base systems*, volume 79, pages 1–14, 1979.

[145] Alexa VanHattum, Rachit Nigam, Vincent T Lee, James Bornholt, and Adrian Sampson. Vectorization for digital signal processors via equality saturation. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 874–886, 2021.

[146] Anthony Vannelli. An adaptation of the interior point method for solving the global routing problem. *IEEE transactions on computer-aided design of integrated circuits and systems*, 10(2):193–203, 2002.

[147] Jelena Vucinic, David Simoncini, Manon Ruffini, Sophie Barbe, and Thomas Schiex. Positive multistate protein design. *Bioinformatics*, 36(1):122–130, 2020.

[148] Gang Wang, Wenrui Gong, Brian DeRenzi, and Ryan Kastner. Ant colony optimizations for resource- and timing-constrained operation scheduling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2007.

[149] Yingheng Wang, Zichen Wang, Gil Sadeh, Luca Zancato, Alessandro Achille, George Karypis, and Huzefa Rangwala. Long-context protein language model. *bioRxiv*, pages 2024–10, 2024.

[150] Yisu Remy Wang, Shana Hutchison, Jonathan Leang, Bill Howe, and Dan Suciu. Spores: Sum-product optimization via relational equality saturation for large scale linear algebra. *Int'l Conf. on Very Large Data Bases (VLDB)*, 13(11), 2020.

[151] Joseph L Watson, David Juergens, Nathaniel R Bennett, Brian L Trippe, Jason Yim, Helen E Eisenach, Woody Ahern, Andrew J Borst, Robert J Ragotte, Lukas F Milles, et al. De novo design of protein structure and function with rfdiffusion. *Nature*, 620(7976):1089–1100, 2023.

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

[153] Ellen M Wijsman. The role of large pedigrees in an era of high-throughput sequencing. *Human genetics*, 131:1555–1563, 2012.

[154] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. egg: Fast and extensible equality saturation. *Proc. ACM Program. Lang.*, 2021.

[155] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[156] Zhiyong Wu, Yaoxiang Wang, Jiacheng Ye, and Lingpeng Kong. Self-adaptive in-context learning: An information compression perspective for in-context example selection and ordering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023.

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

[158] Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. Self-evaluation guided beam search for reasoning. *Advances in Neural Information Processing Systems*, 36:41618–41650, 2023.

[159] AMD Xilinx. Vitis high-level synthesis (hls), 2025. `https://www.amd.com/de/products/software/adaptive-socs-and-fpgas/vitis/vitis-hls.html`.

[160] Yassine Yaakoubi, François Soumis, and Simon Lacoste-Julien. Machine learning in airline crew pairing to construct initial clusters for dynamic constraint aggregation. *EURO Journal on Transportation and Logistics*, 9(4):100020, 2020.

[161] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[162] Yichen Yang, Phitchaya Phothilimthana, Yisu Wang, Max Willsey, Sudip Roy, and Jacques Pienaar. Equality saturation for tensor graph superoptimization. *Conf. on Machine Learning and Systems (MLSys)*, 3:255–268, 2021.

[163] Chen Yanover, Menachem Fromer, and Julia M Shifman. Dead-end elimination for multistate protein design. *Journal of Computational Chemistry*, 28(13):2122–2129, 2007.

[164] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. {τ}-bench: A benchmark for \underline{T}ool-\underline{A}gent-\underline{U}ser interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[165] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[166] Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[167] Kaizhi Yue and Ken A Dill. Forces of tertiary structural organization in globular proteins. *Proceedings of the National Academy of Sciences*, 92(1):146–150, 1995.

[168] Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Zhihan Guo, Yufei Wang, Irwin King, Xue Liu, and Chen Ma. What, how, where, and how well? a survey on test-time scaling in large language models. *arXiv preprint arXiv:2503.24235*, 2025.

[169] Xuanchang Zhang, Wei Xiong, Lichang Chen, Tianyi Zhou, Heng Huang, and Tong Zhang. From lists to emojis: How format bias affects model alignment. *arXiv preprint arXiv:2409.11704*, 2024.

[170] Yihong Zhang. The e-graph extraction problem is np-complete. https://effect.systems/blog/egraph-extraction.html.

[171] Wenting Zhao, Nan Jiang, Celine Lee, Justin T Chiu, Claire Cardie, Matthias Gallé, and Alexander M Rush. Commit0: Library generation from scratch. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[172] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

[173] Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*, 2024.

[174] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[175] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P Xing, et al. Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 559–578, 2022.

[176] Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in llm-based automatic heuristic design. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

[177] Terry Yue Zhuo, Vu Minh Chien, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen GONG, James Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

A Prompt Design

In this section, we detail the system and user prompts used by the LLM agent, as well as the auxiliary prompt employed to enhance our problem descriptions.

A.1 System Prompt

Each iteration of our benchmark begins with a task-agnostic system prompt that instructs the LLM to generate and iteratively refine executable heuristics for combinatorial optimization problems. This system prompt is followed by a task-specific problem statement and an input/output specification. The prompt includes placeholders – highlighted in red – that are dynamically instantiated at runtime for each task. For instance, {NUM_CPU_CORES} represents the CPU core limit for the task (default: 8), and {TIMEOUT} specifies the wall-clock time limit (default: 10 seconds).

System Prompt

You are a world-class optimization expert and algorithmic problem solver. Your task is to develop a highly efficient solution to the following optimization problem. Please analyze the problem background, mathematical formulation, and I/O specifications with extreme rigor and attention to detail.

Your mission is to devise and implement the most performant algorithm possible, optimizing for both computational efficiency and solution quality. You should leverage your deep knowledge of algorithms, data structures, and optimization techniques to craft a powerful solution. You have complete freedom in your algorithmic approach. Think systematically and creatively. Your goal is to push the boundaries of what's possible within the computational constraints. Please strictly follow the instructions below.

1. A problem template is provided below. You only need to implement the solve function. Do NOT modify the function signature including the data types of the input arguments. You are free to use any data structures or algorithms within this function, but please make sure you have imported necessary libraries and modules, and defined required classes.
2. The evaluation machine has {NUM_CPU_CORES} CPU cores and sufficient memory to run your program. The time limit for this question is {TIMEOUT} seconds. You are free to implement parallel algorithms where appropriate to maximize performance.
3. The Python version is 3.12. You may use any standard Python libraries and only the following third-party libraries:
    - numpy==2.2.5
    - networkx==3.4.2
    - pandas==2.2.3
4. Your response should consist of a complete implementation of the 'solve' function. Do NOT include any explanations, comments, additional text, or Markdown formatting.
5. You will receive execution feedback after the user runs your program, including runtime metrics and correctness evaluation.

A.2 User Prompt

For each problem, the first iteration begins with the following user prompt, which introduces the task and its objective to the LLM, along with a program template that the model is expected to complete.

User Prompt

Problem Information
{PROBLEM DESCRIPTION}

Program Template
```
def solve(input_file: str, solution_file: str):
    """
    Solve the optimization problem.

    Please do NOT change the function name and arguments.
    Inputs should be read from input_file
    and outputs should be written to solution_file.
    Input and output formats have been specified in the problem statement.
    """
    raise NotImplementedError(
        "This is a placeholder implementation you need to fill in."
    )
```

A.3 Prompts for Improvement Guidance

Based on the feasibility of the final outputs, we issue one of two improvement prompts in subsequent iterations. If any test cases fail, we provide the following prompt:

Improvement Guidance Case 1

Feedback from Previous Iteration (Iteration `{iteration-1}`)
These are the test cases and results from the previous iteration:
Test Case 1: `{test_name}`
Input File:
{content}
Result:
`{execution_message}`

Test Case 2: `{test_name}`
Input File:
{content}
Result:
`{execution_message}`

...

Improvement Guidance
The program failed to produce valid solutions for some test cases. Please fix the following issues:

1. Check for compilation errors or runtime exceptions.
2. Ensure the program handles all edge cases and meets the problem constraints correctly.
3. Verify that the input and output format match the expected format.
4. Make sure all required functions are implemented correctly, and no external forbidden libraries are used.
5. If the program is not able to produce valid solutions for any test case, please try to find the root cause and fix it.
6. If the program is able to produce valid solutions for some test cases, please try to improve the solution.

Otherwise, if all test cases pass verification, we issue the following prompt:

Improvement Guidance Case 2

Feedback from Previous Iteration (Iteration `{iteration-1}`)
...

Improvement Guidance
Please carefully observe the problem structure and improve upon this program by:

1. Addressing any weaknesses in the previous approach.
2. Introducing more advanced or efficient algorithms.
3. Focusing on improving performance for test cases.

Your goal is to improve the solution for as many test cases as possible, with special attention to those where the previous solution performed poorly.

A.4 Refinement Prompt for Problem Descriptions

To ensure clarity and correctness in problem specification, we employ a human-in-the-loop process. Specifically, we prompt a weaker LLM to flag any unclear or ambiguous statements in the task description. The following prompt is used for this purpose:

Refinement Prompt for Problem Descriptions

If you were to solve the programming task below, do you have any questions? Is there anything I should clarify before you begin writing code?

Problem Description
{PROBLEM DESCRIPTION}

A.5 Example Program Description

The following provides an example problem description for operator scheduling. For other problems, please refer to our GitHub repository.

```
## Background

High-level synthesis (HLS) is an important stage in electronic design automation (EDA), aimed at
↪  translating a high-level program specification (e.g., written in C/C++ or SystemC) into a
↪  cycle-accurate hardware implementation. After the program is parsed and analyzed, it is typically
↪  transformed into an intermediate representation known as a Control Data Flow Graph (CDFG). This
↪  graph captures the operations (e.g., arithmetic, memory accesses) and their control/data
↪  dependencies. The CDFG can further be processed into a Directed Acyclic Graph (DAG) to facilitate
↪  scheduling and optimization.

One of the core challenges in HLS is operator scheduling, which determines the exact control step (or
↪  cycle) at which each operation is executed, while satisfying data dependencies and resource
↪  constraints. Efficient scheduling plays a critical role in optimizing design quality in terms of
↪  performance, area, and power.

## Formalization

Consider a CDFG with $n$ operation nodes $o_i$, where $i \in O = \{1, 2, \ldots, n\}$, and a precedence
↪  relation $\prec$ on $O$ that captures operation dependencies. Each operation $o_i$ is associated
↪  with a cycle delay $d_i \in \mathbb{Z}^+$ and a resource type $r_i \in R = \{1, 2, \ldots, k\}$.
↪  Let $T = \{0, 1, 2, \ldots, L\}$ represent the set of control steps (c-steps), and define a
↪  schedule as an $n$-tuple $s = (t_1, t_2, \ldots, t_n)$, where $t_i \in T$ denotes the start time
↪  (c-step) of operation $o_i$.

A schedule $s$ is feasible if it satisfies all data dependencies:
$\forall i, j \in O: i \prec j \Rightarrow t_i + d_i \leq t_j$.
Let $S$ denote the set of all feasible schedules. For a given schedule $s$, let $N_r(t)$ be the number
↪  of operations that use resource $r$ in control step $t$, and define the total usage of resource $r$
↪  as $N_r = \sum_{t \in T} N_r(t)$.

Given a bound $G_r$ on the number of available instances for each resource type $r \in R$, the operator
↪  scheduling problem is to find a feasible schedule $s \in S$ that minimizes the overall latency $L$,
↪  defined as
```

```
$\min_{s \in S} \max_{i \in O} (t_i + d_i)$,
subject to the resource constraints
$\forall r \in R, t \in T: N_r(t) \leq G_r$.

## Input Format
The input is provided in JSON format with the following structure:

```json
{
  "name": "input",
  "delay": {
    "mul": 3,
    "sub": 1
  },
  "resource": {
    "mul": 2,
    "sub": 1
  },
  "nodes": [
    ["n1", "mul"],
    ["n2", "mul"],
    ["n3", "sub"]
  ],
  "edges": [
    ["n1", "n3", "lhs"],
    ["n2", "n3", "rhs"]
  ]
}
```

Where:
- `name`: Name of the input graph
- `delay`: Maps each resource type to its execution delay in cycles
- `resource`: Maps each resource type to the number of available functional units
- `nodes`: List of nodes, where each node is represented as `[node_id, resource_type]`
- `edges`: List of edges, where each edge is represented as `[source_node, target_node, edge_name]`

Output Format
The output should provide the execution schedule of the program, indicating the start cycle of each
↪ operation. For example, the following output means that `n1` and `n2` start at cycle 0, while `n3`
↪ starts at cycle 3:
```
n1:0
n2:0
n3:3
```
```

B Models

The LLMs used in our experiments are listed in Table 4. All models were accessed via official APIs provided by their respective organizations, except for the Meta models, which are accessed through the OpenRouter [104] API.

Table 5: Model specifications with API names and official pricing

Organization	Model	API Name	Price ($In/$Out)	Type
OpenAI	GPT-o4-mini	o4-mini:high	1.1/4.4	Reasoning
Anthropic	Claude-3.7-Sonnet	claude-3-7-sonnet-20250219	3/15	Reasoning
DeepSeek	DeepSeek-V3	deepseek-chat(0324)	0.27/1.10	Base
DeepSeek	DeepSeek-R1	deepseek-reasoner	0.55/2.19	Reasoning
Google	Gemini-2.5-Flash	gemini-2.5-flash-preview-04-17	0.15/3.5	Reasoning
Google	Gemini-2.5-Pro	gemini-2.5-pro-preview-05-06	1.25/10.0	Reasoning
Meta	LLaMA-3.3-70B	meta-llama/Llama-3.3-70B-Instruct	0.07/0.33	Base
Meta	LLaMA-4-Maverick	meta-llama/Llama-4-Maverick-17B-128E-Instruct	0.27/0.85	Base
Alibaba	Qwen3-235B	qwen3-235b-a22b	0.29/2.86	Reasoning

C Problem Set

In this section, we provide more details on the problems included in Table 2. For a representative problem description used in the prompts, please consult our GitHub repository for additional details.

C.1 Operator Scheduling

Operator scheduling is a critical stage in high-level synthesis (HLS) [31, 107], the process of converting behavioral hardware descriptions into register-transfer level (RTL) implementations. This task involves carefully assigning each operation to a specific clock cycle while managing a variety of constraints such as data dependencies, resource availability, and performance targets. The effectiveness of the scheduling process is vital, as it directly influences key design metrics including area, power consumption, and execution time, making it an important focus in the field of electronic design automation (EDA).

Over the years, researchers have developed a wide range of techniques to tackle the inherent challenges of operator scheduling in HLS. Exact methods, such as those based on integer linear programming (ILP) [61, 105], can provide optimal solutions but often suffer from scalability issues. As a result, many commercial and academic HLS tools [159, 16] rely on heuristics to achieve practical, near-optimal results. Traditional heuristic approaches, including priority-function-based methods [129, 109, 110], focus on balancing resource utilization with performance requirements. Notably, methods leveraging systems of difference constraints (SDC) enable an efficient formulation that captures a rich set of scheduling restrictions and casts the optimization objective into a linear programming (LP) framework [33, 35]. More recently, the incorporation of machine learning techniques [22, 87] has further advanced the state-of-the-art, enhancing both scheduling efficiency and solution quality in the face of increasingly complex hardware designs.

C.2 Technology Mapping

Technology mapping, in the context of logic synthesis for integrated circuits and field-programmable gate arrays (FPGAs), is the process of converting a logic network into an equivalent network of standard cells or logic resources from a specific technology library. The objective is to optimize key design metrics such as area, delay, and power consumption. It is a crucial step in the VLSI design flow and FPGA design flow, determining the actual physical implementation of a design.

Here in our problem setting, we focus on area-optimal technology mapping for lookup table (LUT)-based FPGAs. Given an input logic network, the goal is to cover the network with K-input subgraphs, each of which can be implemented by a K-LUT, while minimizing the number of LUTs representing the circuit area.

The most widely adopted approaches are cut-based methods, which operate in two stages: cut enumeration and cut selection. In this approach, all feasible K-input cuts – i.e., subgraphs with at most K inputs – are enumerated for each node in the boolean network. Then, a dynamic programming-based selection process chooses one cut per node to construct a full LUT cover of the circuit, optimizing for metrics such as area or delay [19, 30, 95]. A refinement of this approach is known as priority cut pruning, which retains only a limited set of the most promising cuts per node rather than considering all possible cuts. This significantly improves scalability for large circuits and is widely implemented in tools such as ABC [14].

C.3 Global Routing

The global routing problem addresses the challenge of planning signal paths across a chip after logic placement, determining how a set of nets should traverse the layout to ensure connectivity while reserving space for detailed routing. Rather than producing exact wire geometries, global routing generates abstract paths through routing regions. This step must account for routing congestion, layer limitations, and timing criticality, while managing a growing number of nets in modern designs like Very-Large-Scale Integration (VLSI). The quality of the global routing solution plays a critical role in determining the feasibility and effectiveness of downstream routing stages and can ultimately dictate the success or failure of physical design closure.

The problem has been studied extensively via sequential and ILP-based methods. Maze routing, introduced by Lee et al. [75], laid the groundwork for sequential approaches, with subsequent improvements such as the work by Soukup [136]. For multi-terminal nets, rectilinear Steiner tree methods were developed [32]. However, sequential routing lacks global coordination and often leads to congestion. ILP-based methods formulate routing as a 0-1 programming, concurrently optimizing over all nets with objectives like wire length and capacity constraints. While exact ILP solvers are computationally intensive, relaxation techniques such as randomized rounding [17] and multi-commodity network flow models [131, 4] have been employed. Interior-point methods for solving the LP relaxation [146, 12] have also proven effective for scalable and near-optimal routing.

Sapatnekar et al. [59] conducted a comprehensive survey on global routing for integrated circuits. Moffitt et al. [96] revisited the problem by briefing the history and emphasizing open challenges that remain unsolved.

C.4 E-Graph Extraction

E-graph [18, 99] is a data structure that compactly represents a set of expressions. Given an input program and a set of rewrite rules, an e-graph is constructed by applying the rules to the program, generating new expressions, and merging equivalent expressions. It has been widely used to represent and explore the huge number of equivalent program space in tensor graph transformation [162, 24], sparse linear algebra optimization [150], code optimization [73, 133], digital signal processor (DSP) compilation [145, 142], circuit datapath synthesis [143, 26], and floating-point arithmetic [108].

In an e-graph, all functionally equivalent terms are organized in the same equivalent classes, known as e-classes. Nodes within each e-class that represent values or operators are called e-nodes. E-classes are a partition of e-nodes, where each e-node belongs to exactly one e-class. Dependencies in e-graphs are directed, which point from e-nodes to their children e-classes, indicating the operator (e-node) requires the values (e-nodes) from the child e-classes to compute its value.

In e-graph extraction, an optimized term from an e-graph is extracted after rewrites, based on a user-defined cost model. The goal is to produce a functionally equivalent but improved implementation of the original input program. The e-graph extraction problem is proven to be NP-hard when common sub-expressions are considered [137, 170].

Existing e-graph extraction methods include exact methods employing ILP [26, 133]. Recently, there has been significant progress in employing heuristics for e-graph extraction. These include a simple working-list method [108], a relaxation method utilizing gradient descent [15], and a specialized method tailored for sparse e-graphs [50]. The dataset used in evaluation for this work primarily comes from SmoothE [15].

C.5 Intra-Operator Parallelism

Intra-Operator Parallelism (IOPDDL), an emerging challenge introduced in the ASPLOS'25 contest track [97], addresses the complexities of distributed deep learning. Leading teams in this competition have predominantly employed meta-heuristic approaches, distinguished by their unique pre-processing and optimization strategies.

The effective distribution of large machine learning models across multiple hardware accelerators is paramount for achieving desired performance in both training and serving applications [175, 172, 130, 113, 76, 44, 23]. This task necessitates sharding the computation graph to minimize communication overhead, a process made intricate by the vast number of operations and tensors involved. Specifically, for a given graph where nodes represent operations with distinct execution strategies (each possessing associated cost and memory usage), an optimal strategy must be chosen for every node. The objective is to minimize the aggregate sum of node and edge costs, without exceeding a strict memory usage constraint across all devices at any point. The inherent diversity in topological and memory characteristics of ML models across varied tasks and modalities renders this problem especially demanding.

C.6 Protein Sequence Design

Understanding how proteins fold into their native three-dimensional structures [67, 151] is a central problem in structural biology [94, 45], traditionally framed as a forward problem: predicting the structure a given amino acid sequence will adopt [82, 149]. In contrast, the protein sequence design or inverse folding problem starts from a fixed target structure and seeks sequences that are likely to fold into it. Many works have shown that this inverse formulation not only offers practical applications in protein engineering but also deepens our understanding of sequence–structure relationships [43, 167, 128, 40, 139, 74].

A common modeling approach treats sequence design as a global optimization problem over the space of amino acid sequences. Methods developed by Sun et al. [139], Shakhnovich and Gutin [128], and others define a fitness function to select sequences with favorable folding properties. These functions are designed to balance positive design (low free energy in the target structure) with negative design (high energy in competing folds), promoting both thermodynamic stability and structural specificity. More recently, people have been working on multi-state design with more or less general fitness functions [112, 7, 5, 98, 163, 54, 147]

In our benchmark, we focus on the Grand Canonical (GC) model [139] of protein sequence design. The GC model operates on (i) a detailed three-dimensional geometric representation of a target structure with n residues, (ii) a simplified binary alphabet distinguishing only hydrophobic (H) and polar (P) residues, and (iii) a fitness function Φ that favors sequences with densely packed hydrophobic cores while penalizing solvent-exposed hydrophobic residues. Despite its simplicity, the H/P model has been shown to capture key qualitative features of real protein structures [41, 68]. Several studies [93, 11] have explored the correspondence between sequences optimized under the GC model and those observed in natural proteins. However, a key obstacle has remained: computing an optimal sequence for a given structure is computationally challenging. The brute-force enumeration over all 2^n H/P sequences is infeasible for realistic protein sizes, and the algorithmic complexity of the problem was explicitly raised as an open question by Hart et al. [56]. An efficient algorithm that constructs an optimal sequence in polynomial runtime was introduced later [70] using network flow.

C.7 Mendelian Error Detection

Chromosomes encode an individual's genetic information, with each gene occupying a specific position known as a locus. At each locus, a diploid organism carries two alleles – one inherited from each parent – forming its genotype. When direct genotyping is not available, researchers rely on the observable traits or phenotypes, which represent sets of compatible genotypes. A group of related individuals, along with their phenotypes at a locus, is organized into a pedigree, where each individual is either a founder or has parents defined within the structure.

Due to experimental and human errors, pedigree data may contain inaccuracies. These errors are classified as either parental errors (incorrect parentage, which we assume do not occur here) or phenotype errors, which can lead to Mendelian errors. A Mendelian error arises when all genotype combinations compatible with observed phenotypes violate Mendel's law that each individual inherits one allele from each parent. Detecting such inconsistencies is computationally challenging; the number of possible genotype combinations grows exponentially with pedigree size, making full enumeration impractical. In fact, verifying consistency has been shown to be NP-complete [1].

Error detection and correction are crucial for downstream tasks like genetic mapping or disease gene localization. However, existing tools are often limited by scalability issues, strong assumptions, or incomplete analysis. To address these limitations, a soft constraint network framework for detecting Mendelian inconsistencies was proposed [123], estimating the minimum number of required corrections, and suggesting optimal modifications. These problems naturally align with weighted constraint satisfaction and provide a rich testbed for scalable and flexible inference in large, complex pedigrees.

C.8 Airline Crew Pairing

The airline crew pairing problem is a well-established topic in operations research. It involves constructing sequences of flight legs – known as pairings – that begin and end at a crew base, cover all scheduled flights, and satisfy a variety of regulatory and contractual constraints. The primary goal is to minimize total crew-related costs, such as wages, hotel accommodations, and deadhead travel, while ensuring legality and operational feasibility. This problem is typically formulated as a set partitioning model and addressed using column generation and branch-and-price techniques [39, 69]. Foundational systems developed for carriers like American Airlines demonstrated the effectiveness of these methods at scale [8]. More recent innovations include dynamic constraint aggregation [48] and machine learning-based pairing generation [160], which are now integral to commercial solvers such as Jeppesen [64] and Sabre [121], capable of processing monthly schedules with tens of thousands of flights.

In addition to exact methods, heuristic and metaheuristic techniques – such as genetic algorithms, simulated annealing, and local search – have been explored to improve scalability and reduce computation time, particularly for medium-sized instances or disruption recovery [89, 135]. These hybrid approaches aim to complement exact optimization methods by leveraging historical data and incorporating planner preferences, offering more flexible and adaptive solutions in practice.

C.9 Pickup and Delivery Problem with Time Windows

The Pick-up and Delivery Problem with Time Windows (PDPTW), originally proposed by Dumas, Desrosiers, and Soumis [47], is generalized from a classical NP-hard combinatorial optimization problem – the Capacitated Vehicle Routing Problem (CVRP). It introduces additional complexity through precedence constraints, requiring pick-up locations to precede corresponding drop-off locations, and service time windows at each location. The problem can be seen in many logistic and public transportation systems, with the primary objective of minimizing the total travel cost.

Over the past three decades, a wide range of models and algorithms have been proposed to address the PDPTW, with most falling into the category of heuristic or metaheuristic approaches. Prominent works include simulated annealing [77, 13], large neighborhood search [34, 120], and iterated local search [124]. In contrast, research into exact solution methods has been relatively limited, with the most effective approaches relying on the set partitioning formulation combined with the branch-cut-and-price algorithm [118, 10]. Ropke et al. [119] provided a comprehensive survey of PDPTW solvers developed up to 2007. Ho et al. [58] later reviewed more recent advancements up to 2018, with a particular emphasis on PDPTW variants for people transportation, referred to as the Dial-a-Ride problem.

To support algorithm development, several benchmark datasets have been created and maintained. The Li and Lim dataset [77] is widely used and includes instances ranging from 100 to 1000 locations. More recently, Sartori and Buriol [124] released a larger-scale dataset generated from real-world spatial-temporal distributions.

D Additional Experiments

In this section, we provide more experimental results and analysis on our benchmark.

D.1 Experimental Settings

By default, we constrain LLMs to generate Python code for each problem and execute the code on a CPU server, with each instance allocated 8 CPU cores. The timeout for each problem is specified in Table 6.

Table 6: Timeout for each problem.

Problem	**Timeout (sec)**
Operator scheduling	10
Technology mapping	10
Global routing	300
E-graph extraction	10
Intra-op parallelism	60
Protein sequence design	10
Mendelian error detection	10
Airline crew pairing	10
Pickup and delivery w/ time windows	60

D.2 Detailed Results on Each Problem

We provide the detailed $\texttt{solve}_s@i$ values for each problem in Tables 7 through 15. The variation in $\texttt{solve}_s@i$ across different problems highlights the diverse levels of difficulty, as summarized in Table 2. For instance, the global routing problem remains unsolved by all evaluated LLMs – even for generating a single feasible solution. In the case of the pickup and delivery problem, the low

$\texttt{solve}_{\text{III}}$@10 ratio also indicates that current LLMs struggle to consistently satisfy the problem's constraints.

Table 7: $\texttt{solve}_s$@i results on operator scheduling problem.

	$\texttt{solve}_{\text{III}}$			$\texttt{solve}_{\text{II}}$			$\texttt{solve}_{\text{I}}$		
Model	@10	@5	@1	@10	@5	@1	@10	@5	@1
DeepSeek-V3	**100.0%**	**100.0%**	4.2%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
DeepSeek-R1	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Gemini-2.5-Flash	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%
Gemini-2.5-Pro	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
LLaMA-4-Maverick	20.8%	0.0%	0.0%	**100.0%**	4.2%	0.0%	**100.0%**	**100.0%**	4.2%
LLaMA-3.3-70B	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Qwen3-235B	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Claude-3.7-Sonnet	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%
GPT-o4-mini	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**

Table 8: $\texttt{solve}_s$@i results on technology mapping problem.

	$\texttt{solve}_{\text{III}}$			$\texttt{solve}_{\text{II}}$			$\texttt{solve}_{\text{I}}$		
Model	@10	@5	@1	@10	@5	@1	@10	@5	@1
DeepSeek-V3	0.0%	0.0%	0.0%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
DeepSeek-R1	87.1%	87.1%	**77.4%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Gemini-2.5-Flash	0.0%	0.0%	0.0%	93.5%	77.4%	67.7%	**100.0%**	**100.0%**	**100.0%**
Gemini-2.5-Pro	74.2%	74.2%	0.0%	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%
LLaMA-4-Maverick	0.0%	0.0%	0.0%	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%
LLaMA-3.3-70B	0.0%	0.0%	0.0%	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	6.5%
Qwen3-235B	0.0%	0.0%	0.0%	**100.0%**	87.1%	0.0%	**100.0%**	**100.0%**	3.2%
Claude-3.7-Sonnet	87.1%	87.1%	0.0%	**100.0%**	**100.0%**	64.5%	**100.0%**	**100.0%**	**100.0%**
GPT-o4-mini	**100.0%**	**100.0%**	45.2%	**100.0%**	**100.0%**	51.6%	**100.0%**	**100.0%**	**100.0%**

Table 9: $\texttt{solve}_s$@i results on global routing problem.

	$\texttt{solve}_{\text{III}}$			$\texttt{solve}_{\text{II}}$			$\texttt{solve}_{\text{I}}$		
Model	@10	@5	@1	@10	@5	@1	@10	@5	@1
DeepSeek-V3	**0.0%**	**0.0%**	**0.0%**	33.3%	33.3%	0.0%	**100.0%**	**100.0%**	**100.0%**
DeepSeek-R1	**0.0%**	**0.0%**	**0.0%**	0.0%	0.0%	0.0%	**100.0%**	**100.0%**	**100.0%**
Gemini-2.5-Flash	**0.0%**	**0.0%**	**0.0%**	20.8%	0.0%	0.0%	**100.0%**	**100.0%**	**100.0%**
Gemini-2.5-Pro	**0.0%**	**0.0%**	**0.0%**	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%
LLaMA-4-Maverick	**0.0%**	**0.0%**	**0.0%**	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
LLaMA-3.3-70B	**0.0%**	**0.0%**	**0.0%**	0.0%	0.0%	0.0%	**100.0%**	**100.0%**	4.2%
Qwen3-235B	**0.0%**	**0.0%**	**0.0%**	0.0%	0.0%	0.0%	**100.0%**	**100.0%**	**100.0%**
Claude-3.7-Sonnet	**0.0%**	**0.0%**	**0.0%**	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%
GPT-o4-mini	**0.0%**	**0.0%**	**0.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**

Table 10: $\texttt{solve}_s$@i results on e-graph extraction problem.

	$\texttt{solve}_{\text{III}}$			$\texttt{solve}_{\text{II}}$			$\texttt{solve}_{\text{I}}$		
Model	@10	@5	@1	@10	@5	@1	@10	@5	@1
DeepSeek-V3	4.3%	0.0%	0.0%	**100.0%**	**100.0%**	82.6%	**100.0%**	**100.0%**	**100.0%**
DeepSeek-R1	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Gemini-2.5-Flash	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%
Gemini-2.5-Pro	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
LLaMA-4-Maverick	0.0%	0.0%	0.0%	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%
LLaMA-3.3-70B	39.1%	39.1%	0.0%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Qwen3-235B	87.0%	87.0%	87.0%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Claude-3.7-Sonnet	39.1%	39.1%	0.0%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
GPT-o4-mini	**100.0%**	**100.0%**	39.1%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**

Table 11: $\texttt{solve}_s@i$ results on intra-op parallelism problem.

Model	$\texttt{solve}_{\text{III}}$ @10	@5	@1	$\texttt{solve}_{\text{II}}$ @10	@5	@1	$\texttt{solve}_{\text{I}}$ @10	@5	@1
DeepSeek-V3	82.1%	53.6%	35.7%	82.1%	53.6%	35.7%	**100.0%**	**100.0%**	**100.0%**
DeepSeek-R1	92.9%	92.9%	35.7%	92.9%	92.9%	35.7%	**100.0%**	**100.0%**	35.7%
Gemini-2.5-Flash	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Gemini-2.5-Pro	82.1%	82.1%	0.0%	82.1%	82.1%	0.0%	**100.0%**	**100.0%**	0.0%
LLaMA-4-Maverick	96.4%	96.4%	3.6%	**100.0%**	**100.0%**	3.6%	**100.0%**	**100.0%**	3.6%
LLaMA-3.3-70B	75.0%	75.0%	3.6%	82.1%	82.1%	3.6%	**100.0%**	**100.0%**	**100.0%**
Qwen3-235B	75.0%	71.4%	67.9%	78.6%	75.0%	75.0%	**100.0%**	**100.0%**	**100.0%**
Claude-3.7-Sonnet	82.1%	82.1%	71.4%	82.1%	82.1%	78.6%	**100.0%**	**100.0%**	96.4%
GPT-o4-mini	**100.0%**	**100.0%**	92.9%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**

Table 12: $\texttt{solve}_s@i$ results on protein sequence design problem.

Model	$\texttt{solve}_{\text{III}}$ @10	@5	@1	$\texttt{solve}_{\text{II}}$ @10	@5	@1	$\texttt{solve}_{\text{I}}$ @10	@5	@1
DeepSeek-V3	83.3%	83.3%	83.3%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
DeepSeek-R1	87.5%	87.5%	0.0%	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%
Gemini-2.5-Flash	95.8%	**95.8%**	**95.8%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Gemini-2.5-Pro	**100.0%**	**95.8%**	0.0%	**100.0%**	95.8%	0.0%	**100.0%**	**100.0%**	4.2%
LLaMA-4-Maverick	83.3%	83.3%	0.0%	95.8%	95.8%	0.0%	**100.0%**	**100.0%**	4.2%
LLaMA-3.3-70B	12.5%	12.5%	12.5%	95.8%	95.8%	95.8%	95.8%	95.8%	95.8%
Qwen3-235B	87.5%	87.5%	87.5%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Claude-3.7-Sonnet	58.3%	45.8%	0.0%	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%
GPT-o4-mini	91.7%	91.7%	91.7%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**

Table 13: $\texttt{solve}_s@i$ results on mendelian error detection problem.

Model	$\texttt{solve}_{\text{III}}$ @10	@5	@1	$\texttt{solve}_{\text{II}}$ @10	@5	@1	$\texttt{solve}_{\text{I}}$ @10	@5	@1
DeepSeek-V3	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%
DeepSeek-R1	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%
Gemini-2.5-Flash	**100.0%**	10.0%	10.0%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Gemini-2.5-Pro	80.0%	80.0%	**80.0%**	80.0%	80.0%	80.0%	**100.0%**	**100.0%**	**100.0%**
LLaMA-4-Maverick	60.0%	60.0%	60.0%	60.0%	60.0%	60.0%	60.0%	60.0%	60.0%
LLaMA-3.3-70B	55.0%	55.0%	55.0%	55.0%	55.0%	55.0%	**100.0%**	**100.0%**	**100.0%**
Qwen3-235B	55.0%	55.0%	0.0%	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%
Claude-3.7-Sonnet	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
GPT-o4-mini	**100.0%**	50.0%	35.0%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**

Table 14: $\texttt{solve}_s@i$ results on airline crew pairing problem.

Model	$\texttt{solve}_{\text{III}}$ @10	@5	@1	$\texttt{solve}_{\text{II}}$ @10	@5	@1	$\texttt{solve}_{\text{I}}$ @10	@5	@1
DeepSeek-V3	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
DeepSeek-R1	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Gemini-2.5-Flash	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	**100.0%**	**100.0%**	14.3%
Gemini-2.5-Pro	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	**100.0%**	**100.0%**	**100.0%**
LLaMA-4-Maverick	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	35.7%	**100.0%**	**100.0%**	**100.0%**
LLaMA-3.3-70B	42.9%	42.9%	42.9%	42.9%	42.9%	42.9%	**100.0%**	**100.0%**	**100.0%**
Qwen3-235B	21.4%	21.4%	0.0%	**100.0%**	85.7%	0.0%	**100.0%**	**100.0%**	0.0%
Claude-3.7-Sonnet	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%
GPT-o4-mini	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**

Table 15: $\texttt{solve}_s@i$ results on pickup and delivery with time windows problem.

Model	$\texttt{solve}_{\text{III}}$ @10	@5	@1	$\texttt{solve}_{\text{II}}$ @10	@5	@1	$\texttt{solve}_{\text{I}}$ @10	@5	@1
DeepSeek-V3	0.0%	0.0%	0.0%	80.0%	73.3%	73.3%	**100.0%**	**100.0%**	**100.0%**
DeepSeek-R1	16.7%	13.3%	3.3%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Gemini-2.5-Flash	**96.7%**	**90.0%**	6.7%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Gemini-2.5-Pro	30.0%	26.7%	**13.3%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
LLaMA-4-Maverick	0.0%	0.0%	0.0%	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%
LLaMA-3.3-70B	0.0%	0.0%	0.0%	**100.0%**	**100.0%**	0.0%	**100.0%**	**100.0%**	0.0%
Qwen3-235B	0.0%	0.0%	0.0%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Claude-3.7-Sonnet	0.0%	0.0%	0.0%	**100.0%**	**100.0%**	16.7%	**100.0%**	**100.0%**	**100.0%**
GPT-o4-mini	3.3%	0.0%	0.0%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**

D.3 Ablation on Temperature

We evaluate various models across different temperature settings, $T \in \{0.0, 0.5, 1.0\}$. For each model, we run 10 iterations per problem and report the highest QYI achieved across these iterations as the final QYI score for that problem. The overall benchmark score is then computed as the arithmetic mean of QYI across all problems. Detailed results are shown in Tables 16 to 18.

In general, improving the temperature can be beneficial to quality as the model becomes more creative, but may harm yield as it may not follow the constraints strictly. Note that yield emphasizes the best iteration that achieves the highest QYI, whereas $\texttt{solve}_{\text{III}}$ reflects the cumulative success rate across iterations; therefore, their values may differ. Additionally, the weighted QYI is not the harmonic mean of weighted yield and weighted quality, as it is computed by aggregating metrics across different problems using a weighted approach.

We also report an uncapped version of the weighted QYI metric[2], which better reflects cases where LLM-generated programs outperform expert solutions on certain test instances. Improvements are underlined in the tables. While this variant achieves slightly higher scores for most models – indicating occasional superior performance – it also confirms that, in the majority of cases, LLMs still lag significantly behind expert solutions.

Table 16: Performance of different models on Temperature $= 0$.

Model	Weighted Yield	Weighted Quality	Weighted QYI (Capped)	Weighted QYI (Uncapped)
`Claude-3.7-Sonnet`	0.5963	0.4686	0.5034	0.5034
`DeepSeek-R1`	**0.6972**	0.5775	0.5498	<u>0.5553</u>
`DeepSeek-V3`	0.4587	0.3890	0.3707	0.3707
`Gemini-2.5-Flash`	0.6606	0.5281	0.5682	<u>0.5753</u>
`Gemini-2.5-Pro`	0.6468	**0.6700**	**0.6170**	<u>**0.6228**</u>
`LLaMA-3.3-70B`	0.3394	0.3521	0.2951	0.2953
`LLaMA-4-Maverick`	0.3211	0.3383	0.2955	0.2955
`Qwen3-235B`	0.4450	0.4513	0.4355	<u>0.4423</u>

Table 17: Performance of different models on Temperature $= 0.5$.

Model	Weighted Yield	Weighted Quality	Weighted QYI (Capped)	Weighted QYI (Uncapped)
`Claude-3.7-Sonnet`	**0.6147**	**0.6468**	**0.5437**	<u>**0.5451**</u>
`DeepSeek-R1`	0.5138	0.5751	0.4743	<u>0.4812</u>
`DeepSeek-V3`	0.3716	0.4645	0.3322	0.3322
`Gemini-2.5-Flash`	0.4817	0.5700	0.4760	<u>0.4828</u>
`Gemini-2.5-Pro`	0.4817	0.5609	0.4767	<u>0.4789</u>
`LLaMA-3.3-70B`	0.3991	0.4407	0.4108	0.4108
`LLaMA-4-Maverick`	0.3349	0.3712	0.3050	<u>0.3646</u>
`Qwen3-235B`	0.4128	0.4798	0.4269	<u>0.4327</u>

D.4 Few-Shot Demonstration

Table 19 highlights the impact of few-shot demonstrations on LLM performance across the entire HeuriGym benchmark. Introducing only a small number of demonstrations (e.g., three) can negatively affect solution quality and success rate, as these examples may not be representative of the overall dataset, leading the model to overfit to them. However, providing a larger set of demonstrations can potentially improve QYI, as the model benefits from greater diversity and can learn more generalizable patterns.

[2] The uncapped version of quality is computed as $1/\hat{N} \sum_{n=1}^{\hat{N}} c_n^\star / c_n$, and the uncapped QYI is derived by substituting the original quality metric with this uncapped variant.

Table 18: Performance of different models on Temperature $= 1$.

Model	Weighted Yield	Weighted Quality	Weighted QYI (Capped)	Weighted QYI (Uncapped)
`Claude-3.7-Sonnet`	0.5138	0.5924	0.4828	0.4841
`DeepSeek-R1`	0.5688	0.5625	0.5313	0.5383
`DeepSeek-V3`	0.4128	0.4188	0.3839	0.3841
`GPT-o4-mini`	**0.6927**	0.6440	**0.6089**	**0.6158**
`Gemini-2.5-Flash`	0.4771	**0.7688**	0.5030	0.5047
`Gemini-2.5-Pro`	0.5229	0.4893	0.4921	0.4981
`LLaMA-3.3-70B`	0.3028	0.3627	0.2868	0.2916
`LLaMA-4-Maverick`	0.2982	0.3271	0.2667	0.2672
`Qwen3-235B`	0.5459	0.5228	0.5294	0.5364

Table 19: Impact of few-shot demonstrations on performance (Model: `Gemini-2.5-pro`).

# of Demos	Weighted Yield	Weighted Quality	Weighted QYI
Zero-shot	0.5872	0.7159	0.5999
Half-shot	0.5092	0.6526	0.5361
Full-shot	0.6468	0.6700	0.6170

D.5 Feedback Rounds

Table 20 shows that increasing the number of feedback rounds has a nuanced impact on performance. While a moderate number of rounds (e.g., five) can enhance overall quality by guiding the model to refine its solutions, excessive feedback may lead to diminishing returns or even degrade performance. This suggests that too many rounds can overwhelm the model, making it harder to identify and prioritize the most critical information from the feedback.

Table 20: Impact of feedback rounds on performance (Model: `Gemini-2.5-pro`).

# of Feedback Rounds	Weighted Yield	Weighted Quality	Weighted QYI
1	0.6193	0.7290	0.6253
5	0.6055	0.7313	0.6259
10	0.6468	0.6700	0.6170

D.6 Iterative Best-of-N Sampling

To investigate the benefits of test-time search strategies, we sample k candidate programs in each iteration, evaluate them, and return feedback for all k programs to the LLM. After a fixed number of iterations, we select the best-performing program from the entire pool – a process we refer to as *iterative best-of-N sampling*. The total number of sampled programs is held constant across different values of k. This strategy allows the model to explore diverse candidate solutions in parallel and evolve the program based on evaluative feedback.

As shown in Table 21, increasing k leads to better quality of results, indicating that aggregating feedback across multiple candidates allows the LLM to better explore the solution space and improve sampling efficiency by allocating computational budget toward more informative evaluations.

Table 21: Impact of best-of-N sampling on performance (Model: `Gemini-2.5-pro`).

# of Samples @ Iteration	Weighted Yield	Weighted Quality	Weighted QYI
2@5	0.5688	0.7698	0.6160
1@10	0.6468	0.6700	0.6170

D.7 Error Analysis

In the following, we present representative examples of common errors made by LLMs during heuristic generation. These errors highlight current limitations in code reliability and execution:

- **Import error:** This type of error occurs when the generated code relies on external libraries that are not available in the environment. In the example below, the model attempts to import the `ortools` library, which results in a `ModuleNotFoundError`. Such errors suggest that the model does not strictly follow the instructions given in the prompt.

```
  File "operator_scheduling/gemini-2.5-flash-preview-04-17/iteration4/sol
  ↪  ver.py", line 2, in
  ↪  <module>
    from ortools.sat.python import cp_model
ModuleNotFoundError: No module named 'ortools'
```

- **API misuse error:** LLMs often misuse APIs due to a misunderstanding of library interfaces. In the following case, the model tries to call `random()` directly from the `random` module, which is not callable.

```
  File "intra_op_parallel/o4-mini/iteration3/solver.py", line 64, in
  ↪  init_jitter
    if len(ci) > 1 and random() < 0.1:
                       ^^^^^^^^
TypeError: 'module' object is not callable
```

- **Syntax error:** Syntax errors are common when the model fails to adhere to basic language rules. In this example, there is an unmatched parenthesis in a `while` loop condition, leading to a `SyntaxError`. Such mistakes typically indicate a lack of code completion validation in the generation process.

```
  File "crew_pairing/deepseek-chat/iteration7/solver.py", line 60
    while len(used_legs) < len(df)):
                                  ^
SyntaxError: unmatched ')'
```

- **Runtime error:** Even syntactically and semantically correct code can fail at runtime. In this case, the model modifies a dictionary while iterating over it, which raises a `RuntimeError`. This highlights the model's difficulty in reasoning about the actual executable code in a long context.

```
  File "technology_mapping/llama-4-maverick/iteration2/solver.py", line
  ↪  104, in technology_mapping
    for successor in G.successors(node):
RuntimeError: dictionary changed size during iteration
```

D.8 C++ Example

We conduct preliminary experiments on the technology mapping problem by modifying the prompt to instruct the LLM to generate a C++ solution, using the provided function template: `void solve(const std::string& input_file, const std::string& output_file)`.

Integrating C++ into our agentic feedback loop remains challenging due to dependencies on domain-specific libraries and the complexity of parallel execution. As a result, our preliminary experiment with C++ involves only a single iteration of prompting.

Table 22 presents a performance comparison between the Python solution with 10 iterations and the C++ solution with just one iteration. Although the C++ solution does not produce high-quality output in its initial attempt, it already achieves a better yield than the Python solution after 10 iterations – an unexpectedly strong outcome. Notably, the Python solution fails to generate any valid result in its first iteration. This is attributed to the significantly faster execution speed of C++ code, which enables it to avoid the timeout errors frequently encountered by Python in this task.

We expect to see further performance improvement with C++ after we integrate it into the feedback loop in our framework.

Table 22: Impact of C++ code on technology mapping performance (Model: `Gemini-2.5-pro`).

Language	# of Iterations	Yield	Quality	QYI
Python	10	0.7419	0.6423	0.6885
C++	1	0.7742	0.3493	0.4814

E Detailed Analysis of Case Study

Here we include a detailed analysis of the case study in Section 5.3. We append the entire code of each iteration and analyze the generated solution.

In the first iteration, the `GPT-o4-mini` model generates an algorithm based on cut enumeration and dynamic programming (DP)-based cut selection, which is the common approach used in technology mapping tools. The core logic is highlighted in the following code snippet. However, it does not prune the cut enumeration space and computes all possible K-feasible cuts for each node, which leads to an exponential explosion in computation time. While this achieves high quality, it frequently triggers `TimeoutError` for larger datasets, resulting in low yield.

```python
# iteration 1: base algorithm without pruning the search space

def solve(input_file: str, solution_file: str):
    from collections import deque
    class Node:
        __slots__ = ('name','fanin_names','fanins','fanouts','patterns_one','patterns_zero','const_val')
        def __init__(self,name):
            self.name = name
            self.fanin_names = []
            self.fanins = []
            self.fanouts = []
            self.patterns_one = []
            self.patterns_zero = []
            self.const_val = None
    raw_lines = open(input_file).read().splitlines()
    lines = []
    i = 0
    while i < len(raw_lines):
        line = raw_lines[i].rstrip()
        if not line or line.lstrip().startswith('#'):
            i += 1
            continue
        if line.endswith('\\'):
            merged = line[:-1].rstrip()
            i += 1
            while i < len(raw_lines):
                nxt = raw_lines[i].rstrip()
                if nxt.endswith('\\'):
                    merged += ' ' + nxt[:-1].rstrip()
                    i += 1
                else:
                    merged += ' ' + nxt
                    i += 1
                    break
            lines.append(merged)
        else:
            lines.append(line)
            i += 1
    model_name = ''
    input_list = []
    output_list = []
    nodes = {}
    idx = 0
    while idx < len(lines):
        l = lines[idx].strip()
        if not l or l.startswith('#'):
            idx += 1
            continue
        if l.startswith('.model'):
            parts = l.split()
            if len(parts) > 1:
                model_name = parts[1]
            idx += 1
        elif l.startswith('.inputs'):
            parts = l.split()
            for name in parts[1:]:
```

```python
                input_list.append(name)
                if name not in nodes:
                    nodes[name] = Node(name)
            idx += 1
        elif l.startswith('.outputs'):
            parts = l.split()
            for name in parts[1:]:
                output_list.append(name)
            idx += 1
        elif l.startswith('.names'):
            parts = l.split()
            in_names = parts[1:-1]
            out_name = parts[-1]
            if out_name not in nodes:
                nodes[out_name] = Node(out_name)
            node = nodes[out_name]
            node.fanin_names = in_names[:]
            patterns = []
            j = idx + 1
            while j < len(lines):
                lj = lines[j].strip()
                if not lj or lj.startswith('.') or lj.startswith('#'):
                    break
                patterns.append(lj)
                j += 1
            if len(in_names) == 0:
                if patterns and patterns[0].strip() == '1':
                    node.const_val = 1
                else:
                    node.const_val = 0
            else:
                for pline in patterns:
                    parts2 = pline.split()
                    if len(parts2) == 1:
                        mask = parts2[0]; bit = '1'
                    else:
                        mask = parts2[0]; bit = parts2[1]
                    if bit == '1':
                        node.patterns_one.append(mask)
                    else:
                        node.patterns_zero.append(mask)
            idx = j
        elif l.startswith('.end'):
            break
        else:
            idx += 1
    for node in nodes.values():
        node.fanins = []
        for fn in node.fanin_names:
            if fn in nodes:
                node.fanins.append(nodes[fn])
        node.fanouts = []
    for node in nodes.values():
        for f in node.fanins:
            f.fanouts.append(node)
    reachable = set()
    dq = deque()
    for out in output_list:
        if out in nodes:
            dq.append(out)
    while dq:
        u = dq.pop()
        if u in reachable:
            continue
        reachable.add(u)
        for f in nodes[u].fanins:
            if f.name not in reachable:
                dq.append(f.name)
    nodes = {name:node for name,node in nodes.items() if name in reachable}
    for node in nodes.values():
        node.fanins = [f for f in node.fanins if f.name in nodes]
        node.fanouts = [f for f in node.fanouts if f.name in nodes]
    indeg = {name: len(node.fanins) for name,node in nodes.items()}
    dq = deque([name for name,d in indeg.items() if d == 0])
    topo_names = []
    while dq:
        u = dq.popleft()
        topo_names.append(u)
        for w in nodes[u].fanouts:
            indeg[w.name] -= 1
            if indeg[w.name] == 0:
```

```
                dq.append(w.name)
    topo_list = [nodes[name] for name in topo_names]
    K = 6
    def prune_cuts(cset):
        cuts = list(cset)
        res = []
        for c in cuts:
            skip = False
            for d in cuts:
                if d is not c and d.issubset(c):
                    skip = True
                    break
            if not skip:
                res.append(c)
        return res
    cuts = {}      # Cut Enumeration
    for n in topo_list:
        if not n.fanins:
            cuts[n.name] = [frozenset([n.name])]
        else:
            cuts_n = None
            for f in n.fanins:
                cf = cuts[f.name]
                if cuts_n is None:
                    cuts_n = cf[:]
                else:
                    newset = set()
                    for c1 in cuts_n:
                        for c2 in cf:
                            u = c1 | c2
                            if len(u) <= K:
                                newset.add(u)
                    cuts_n = prune_cuts(newset)
            if cuts_n is None:
                cuts_n = []
            cuts_set = set(cuts_n)
            cuts_set.add(frozenset([n.name]))
            cuts[n.name] = prune_cuts(cuts_set)
    cost = {}
    best_cut = {}
    INF = 10**18
    for n in topo_list:    # DP-based Cut Selection
        if not n.fanins:
            cost[n.name] = 0
            best_cut[n.name] = None
        else:
            bcut = None
            bval = INF
            for c in cuts[n.name]:
                if c == frozenset([n.name]):
                    continue
                s = 0
                for m in c:
                    s += cost.get(m,0)
                cval = s + 1
                if cval < bval:
                    bval = cval; bcut = c
            if bcut is None:
                fl = frozenset([f.name for f in n.fanins])
                bcut = fl
                s = 0
                for m in bcut:
                    s += cost.get(m,0)
                bval = s + 1
            cost[n.name] = bval
            best_cut[n.name] = bcut
    mapping_nodes = set()
    dq = deque()
    for out in output_list:
        if out in best_cut and best_cut[out] is not None:
            dq.append(out)
    while dq:
        u = dq.pop()
        if u in mapping_nodes:
            continue
        mapping_nodes.add(u)
        c = best_cut[u]
        if c:
            for m in c:
                if m in nodes:
                    nm = nodes[m]
```

```
                    if m not in input_list and nm.const_val is None:
                        dq.append(m)
    mapping_topo = [n for n in topo_list if n.name in mapping_nodes]
    f = open(solution_file,'w')
    f.write('.model '+model_name+'\n')
    f.write('.inputs '+' '.join(input_list)+'\n')
    f.write('.outputs '+' '.join(output_list)+'\n')
    for n in mapping_topo:
        leaves = best_cut[n.name]
        inputs = sorted(leaves)
        k = len(inputs)
        N = 1<<k
        full_mask = (1<<N) - 1
        masks = []
        for i in range(k):
            m = 0
            for j in range(N):
                if (j>>i)&1:
                    m |= (1<<j)
            masks.append(m)
        vis = set()
        queue2 = [n.name]
        while queue2:
            u = queue2.pop()
            if u in vis or u in inputs:
                continue
            vis.add(u)
            for fn in nodes[u].fanins:
                if fn.name not in vis:
                    queue2.append(fn.name)
        cone_nodes = [nodes[name] for name in topo_names if name in vis]
        val = {}
        for i,name in enumerate(inputs):
            val[name] = masks[i]
        for v in cone_nodes:
            if v.const_val is not None:
                val[v.name] = full_mask if v.const_val==1 else 0
            else:
                if v.patterns_zero:
                    zero_mask = 0
                    for p in v.patterns_zero:
                        m = full_mask
                        for idx,ch in enumerate(p):
                            vm = val[v.fanin_names[idx]]
                            if ch=='1':
                                m &= vm
                            elif ch=='0':
                                m &= (~vm)&full_mask
                        zero_mask |= m
                    val[v.name] = (~zero_mask)&full_mask
                else:
                    one_mask = 0
                    for p in v.patterns_one:
                        m = full_mask
                        for idx,ch in enumerate(p):
                            vm = val[v.fanin_names[idx]]
                            if ch=='1':
                                m &= vm
                            elif ch=='0':
                                m &= (~vm)&full_mask
                        one_mask |= m
                    val[v.name] = one_mask
        root_mask = val[n.name]
        f.write('.names '+' '.join(inputs)+' '+n.name+'\n')
        for j in range(N):
            if (root_mask>>j)&1:
                line = ''.join('1' if (j>>i)&1 else '0' for i in range(k))
                f.write(line+' 1\n')
    f.write('.end\n')
    f.close()
```

In the second iteration, the model overcorrects by adopting a naive 6-LUT mapping strategy that trivially assigns each node to a separate LUT. While this approach passes all test cases within the timeout, it produces poor solution quality. Specifically, it redundantly maps each non-primary-input node as an individual LUT by directly copying inputs to outputs, resulting in excessive and inefficient resource usage.

```python
# iteration 2: a very naive mapping strategy. Simply copy the input to the output.

def solve(input_file: str, solution_file: str):
    raw = open(input_file).read().splitlines()
    merged = []
    i = 0
    while i < len(raw):
        line = raw[i].rstrip()
        if not line or line.lstrip().startswith('#'):
            i += 1
            continue
        if line.endswith('\\'):
            accum = line[:-1].rstrip()
            i += 1
            while i < len(raw):
                nxt = raw[i].rstrip()
                if nxt.endswith('\\'):
                    accum += ' ' + nxt[:-1].rstrip()
                    i += 1
                else:
                    accum += ' ' + nxt
                    i += 1
                    break
            merged.append(accum)
        else:
            merged.append(line)
            i += 1
    with open(solution_file, 'w') as f:
        for l in merged:
            f.write(l + '\n')
```

In the third iteration, the model proposes a more sophisticated strategy that first applies static pruning to the cut enumeration space – limiting the number of cuts per node to 20 – followed by a dynamic programming-based cut selection. This approach improves solution quality, though full yield is still not achieved. Additionally, truth table computation is optimized using bitwise operations.

```python
# iteration 3: static pruning by capping the number of cuts per node to M = 20; efficient bitwise operations

def solve(input_file: str, solution_file: str):
    from collections import deque
    raw_lines = open(input_file).read().splitlines()
    lines = []
    i = 0
    while i < len(raw_lines):
        line = raw_lines[i].rstrip()
        if not line or line.lstrip().startswith('#'):
            i += 1
            continue
        if line.endswith('\\'):
            accum = line[:-1].rstrip()
            i += 1
            while i < len(raw_lines):
                nxt = raw_lines[i].rstrip()
                if nxt.endswith('\\'):
                    accum += ' ' + nxt[:-1].rstrip()
                    i += 1
                else:
                    accum += ' ' + nxt
                    i += 1
                    break
            lines.append(accum)
        else:
            lines.append(line)
            i += 1
    class Node:
        def __init__(self, name):
            self.name = name
            self.fanin_names = []
            self.patterns_one = []
            self.patterns_zero = []
            self.const_val = None
            self.fanins = []
            self.fanouts = []
    model_name = ''
    input_list = []
    output_list = []
```

```python
    nodes = {}
    idx = 0
    while idx < len(lines):
        l = lines[idx].strip()
        if not l:
            idx += 1
            continue
        if l.startswith('.model'):
            parts = l.split()
            if len(parts) > 1:
                model_name = parts[1]
            idx += 1
        elif l.startswith('.inputs'):
            parts = l.split()
            for name in parts[1:]:
                input_list.append(name)
                if name not in nodes:
                    nodes[name] = Node(name)
            idx += 1
        elif l.startswith('.outputs'):
            parts = l.split()
            for name in parts[1:]:
                output_list.append(name)
            idx += 1
        elif l.startswith('.names'):
            parts = l.split()
            in_names = parts[1:-1]
            out_name = parts[-1]
            if out_name not in nodes:
                nodes[out_name] = Node(out_name)
            node = nodes[out_name]
            node.fanin_names = in_names[:]
            patterns = []
            j = idx + 1
            while j < len(lines) and not lines[j].startswith('.'):
                s = lines[j].strip()
                if s:
                    patterns.append(s)
                j += 1
            if len(in_names) == 0:
                if patterns and patterns[0] == '1':
                    node.const_val = 1
                else:
                    node.const_val = 0
            else:
                for pline in patterns:
                    parts2 = pline.split()
                    if len(parts2) == 1:
                        mask = parts2[0]; bit = '1'
                    else:
                        mask = parts2[0]; bit = parts2[1]
                    if bit == '1':
                        node.patterns_one.append(mask)
                    else:
                        node.patterns_zero.append(mask)
            idx = j
        elif l.startswith('.end'):
            break
        else:
            idx += 1
    for node in nodes.values():
        node.fanins = [nodes[nm] for nm in node.fanin_names if nm in nodes]
        node.fanouts = []
    for node in nodes.values():
        for f in node.fanins:
            f.fanouts.append(node)
    reachable = set()
    stack = [nm for nm in output_list if nm in nodes]
    while stack:
        u = stack.pop()
        if u in reachable: continue
        reachable.add(u)
        for p in nodes[u].fanin_names:
            if p in nodes and p not in reachable:
                stack.append(p)
    nodes = {nm:nd for nm,nd in nodes.items() if nm in reachable}
    for nd in nodes.values():
        nd.fanins = [f for f in nd.fanins if f.name in nodes]
        nd.fanouts = [f for f in nd.fanouts if f.name in nodes]
    indeg = {nm: len(nd.fanins) for nm,nd in nodes.items()}
    dq = deque([nm for nm,d in indeg.items() if d == 0])
```

```
    topo = []
    while dq:
        u = dq.popleft()
        topo.append(u)
        for w in nodes[u].fanouts:
            indeg[w.name] -= 1
            if indeg[w.name] == 0:
                dq.append(w.name)
    topo_nodes = [nodes[nm] for nm in topo]
    name_to_idx = {nd.name:i for i,nd in enumerate(topo_nodes)}
    idx_to_node = topo_nodes
    idx_to_name = [nd.name for nd in topo_nodes]
    N = len(topo_nodes)
    PI_idx = set(name_to_idx[nm] for nm in input_list if nm in name_to_idx)
    self_mask = [1 << i for i in range(N)]
    K = 6
    M = 20
    cuts = [[] for _ in range(N)]
    for i, nd in enumerate(topo_nodes):
        if i in PI_idx or nd.const_val is not None:
            cuts[i] = [self_mask[i]]
            continue
        fan_idxs = [name_to_idx[x] for x in nd.fanin_names if x in name_to_idx]
        c_list = None
        for f in fan_idxs:
            fcuts = cuts[f]
            if c_list is None:
                c_list = fcuts[:M]
            else:
                newset = set()
                for a in c_list[:M]:
                    for b in fcuts[:M]:
                        u = a | b
                        if u.bit_count() <= K:
                            newset.add(u)
                if newset:
                    lst = sorted(newset, key=lambda x: x.bit_count())
                    c_list = lst[:M]
                else:
                    c_list = []
            if not c_list:
                break
        if c_list is None:
            c_list = []
        s = set(c_list)
        s.add(self_mask[i])
        lst2 = sorted(s, key=lambda x: x.bit_count())
        cuts[i] = lst2[:M]
    INF = 10**18
    cost = [0]*N
    bestM = [None]*N
    for i, nd in enumerate(topo_nodes):
        if i in PI_idx or nd.const_val is not None:
            cost[i] = 0
            bestM[i] = None
            continue
        bestc = INF
        bm = None
        for cm in cuts[i]:
            if cm == self_mask[i]:
                continue
            s = 0
            m = cm
            while m:
                lsb = m & -m
                j = lsb.bit_length() - 1
                s += cost[j]
                m ^= lsb
            cval = s + 1
            if cval < bestc:
                bestc = cval
                bm = cm
        if bm is None:
            s = 0
            mask = 0
            for f in nd.fanin_names:
                if f in name_to_idx:
                    j = name_to_idx[f]
                    mask |= self_mask[j]
                    s += cost[j]
            bm = mask
```

```
            bestc = s + 1
        cost[i] = bestc
        bestM[i] = bm
    mapping = set()
    stack = [name_to_idx[nm] for nm in output_list if nm in name_to_idx]
    visited = set()
    while stack:
        u = stack.pop()
        if u in visited:
            continue
        visited.add(u)
        bm = bestM[u]
        if bm is None or bm == self_mask[u]:
            continue
        mapping.add(u)
        m = bm
        while m:
            lsb = m & -m
            j = lsb.bit_length() - 1
            m ^= lsb
            if j not in visited and j not in PI_idx and topo_nodes[j].const_val is None:
                stack.append(j)
    mapped = [i for i in range(N) if i in mapping]
    with open(solution_file, 'w') as f:
        f.write('.model ' + model_name + '\n')
        f.write('.inputs ' + ' '.join(input_list) + '\n')
        f.write('.outputs ' + ' '.join(output_list) + '\n')
        for i in mapped:
            nd = topo_nodes[i]
            bm = bestM[i]
            leaves = [j for j in range(N) if (bm >> j) & 1]
            inputs = [idx_to_name[j] for j in leaves]
            k = len(inputs)
            Np = 1 << k
            full = (1 << Np) - 1
            masks = []
            for t in range(k):
                mm = 0
                for j in range(Np):
                    if (j >> t) & 1:
                        mm |= 1 << j
                masks.append(mm)
            val = {}
            for t,j in enumerate(leaves):
                val[j] = masks[t]
            vis = set(leaves)
            stack2 = [i]
            while stack2:
                u = stack2.pop()
                if u in vis:
                    continue
                vis.add(u)
                for p in topo_nodes[u].fanins:
                    pi = name_to_idx.get(p.name)
                    if pi is not None and pi not in vis:
                        stack2.append(pi)
            cone = [j for j in range(N) if j in vis and j not in leaves]
            for j in sorted(cone, key=lambda x: name_to_idx[topo_nodes[x].name]):
                v = topo_nodes[j]
                if v.const_val is not None:
                    val[j] = full if v.const_val == 1 else 0
                else:
                    if v.patterns_zero:
                        zm = 0
                        for p in v.patterns_zero:
                            mm = full
                            for t,ch in enumerate(p):
                                fn = v.fanin_names[t]
                                pj = name_to_idx.get(fn)
                                if pj is None:
                                    mm = 0
                                    break
                                vm = val[pj]
                                if ch == '1':
                                    mm &= vm
                                elif ch == '0':
                                    mm &= (~vm) & full
                            zm |= mm
                        val[j] = (~zm) & full
                    else:
                        om = 0
```

```
                        for p in v.patterns_one:
                            mm = full
                            for t,ch in enumerate(p):
                                fn = v.fanin_names[t]
                                pj = name_to_idx.get(fn)
                                if pj is None:
                                    mm = 0
                                    break
                                vm = val[pj]
                                if ch == '1':
                                    mm &= vm
                                elif ch == '0':
                                    mm &= (~vm) & full
                            om |= mm
                        val[j] = om
            root = val[i]
            f.write('.names ' + ' '.join(inputs) + ' ' + nd.name + '\n')
            for j in range(Np):
                if (root >> j) & 1:
                    bits = ''.join('1' if (j >> t) & 1 else '0' for t in range(k))
                    f.write(bits + ' 1\n')
        f.write('.end\n')

```

In the fourth iteration, the cut limit per node is increased from 20 to 30, enabling broader solution exploration and potentially improving quality. Additionally, the algorithm reduces redundant computations by caching precomputed scores for each cut.

```
# iteration 4: explore larger solution space; reduce redundant computations

def solve(input_file: str, solution_file: str):
    from collections import deque
    raw = open(input_file).read().splitlines()
    lines = []
    i = 0
    while i < len(raw):
        l = raw[i].rstrip()
        if not l or l.lstrip().startswith('#'):
            i += 1
            continue
        if l.endswith('\\'):
            acc = l[:-1].rstrip()
            i += 1
            while i < len(raw):
                nl = raw[i].rstrip()
                if nl.endswith('\\'):
                    acc += ' ' + nl[:-1].rstrip()
                    i += 1
                else:
                    acc += ' ' + nl
                    i += 1
                    break
            lines.append(acc)
        else:
            lines.append(l)
            i += 1
    class Node:
        __slots__ = ('name','fanin_names','patterns_one','patterns_zero','const_val','fanins','fanouts')
        def __init__(self,n):
            self.name = n
            self.fanin_names = []
            self.patterns_one = []
            self.patterns_zero = []
            self.const_val = None
            self.fanins = []
            self.fanouts = []
    model = ''
    inputs = []
    outputs = []
    nodes = {}
    idx = 0
    while idx < len(lines):
        l = lines[idx].strip()
        if not l:
            idx += 1; continue
        if l.startswith('.model'):
            parts = l.split()
            if len(parts)>1: model = parts[1]
```

```
            idx += 1
        elif l.startswith('.inputs'):
            parts = l.split()
            for nm in parts[1:]:
                inputs.append(nm)
                if nm not in nodes: nodes[nm] = Node(nm)
            idx += 1
        elif l.startswith('.outputs'):
            parts = l.split()
            for nm in parts[1:]:
                outputs.append(nm)
            idx += 1
        elif l.startswith('.names'):
            parts = l.split()
            inps = parts[1:-1]; outp = parts[-1]
            if outp not in nodes: nodes[outp] = Node(outp)
            nd = nodes[outp]
            nd.fanin_names = inps[:]
            pats = []
            j = idx+1
            while j < len(lines) and not lines[j].startswith('.'):
                s = lines[j].strip()
                if s: pats.append(s)
                j += 1
            if not inps:
                if pats and pats[0]=='1': nd.const_val = 1
                else: nd.const_val = 0
            else:
                for pt in pats:
                    sp = pt.split()
                    if len(sp)==1:
                        mask = sp[0]; bit = '1'
                    else:
                        mask,bit = sp[0],sp[1]
                    if bit=='1': nd.patterns_one.append(mask)
                    else: nd.patterns_zero.append(mask)
            idx = j
        elif l.startswith('.end'):
            break
        else:
            idx += 1
    for nd in nodes.values():
        nd.fanins = [nodes[nm] for nm in nd.fanin_names if nm in nodes]
    for nd in nodes.values():
        for f in nd.fanins:
            f.fanouts.append(nd)
    reachable = set()
    st = [nm for nm in outputs if nm in nodes]
    while st:
        u = st.pop()
        if u in reachable: continue
        reachable.add(u)
        for p in nodes[u].fanin_names:
            if p in nodes and p not in reachable:
                st.append(p)
    nodes = {nm:nd for nm,nd in nodes.items() if nm in reachable}
    for nd in nodes.values():
        nd.fanins = [f for f in nd.fanins if f.name in nodes]
        nd.fanouts = [f for f in nd.fanouts if f.name in nodes]
    indeg = {nm: len(nd.fanins) for nm,nd in nodes.items()}
    dq = deque([nm for nm,d in indeg.items() if d==0])
    topo = []
    while dq:
        u = dq.popleft(); topo.append(u)
        for w in nodes[u].fanouts:
            indeg[w.name] -= 1
            if indeg[w.name]==0: dq.append(w.name)
    topo_nodes = [nodes[nm] for nm in topo]
    N = len(topo_nodes)
    name_to_idx = {nd.name:i for i,nd in enumerate(topo_nodes)}
    idx_to_name = [nd.name for nd in topo_nodes]
    PI = set(name_to_idx[nm] for nm in inputs if nm in name_to_idx)
    cost = [0]*N
    bestM = [None]*N
    K = 6
    M = 30
    cuts = [[] for _ in range(N)]
    for i,nd in enumerate(topo_nodes):
        if i in PI or nd.const_val is not None:
            cost[i] = 0
            bestM[i] = None
```

```
            cuts[i] = [(1<<i, 0)]
            continue
        fans = [name_to_idx[nm] for nm in nd.fanin_names if nm in name_to_idx]
        c_list = None
        for f in fans:
            fcuts = cuts[f]
            if len(fcuts) > M: fcuts = fcuts[:M]
            if c_list is None:
                c_list = fcuts.copy()
            else:
                newm = {}
                for m1,sc1 in c_list:
                    for m2,sc2 in fcuts:
                        m = m1 | m2
                        if m.bit_count() <= K:
                            if m in newm: continue
                            t = m; sc = 0
                            while t:
                                lsb = t & -t; j = lsb.bit_length()-1
                                sc += cost[j]; t ^= lsb
                            newm[m] = sc
                if not newm:
                    c_list = []
                    break
                items = sorted(newm.items(), key=lambda x: x[1])
                c_list = items[:M]
        if c_list is None: c_list = []
        fb = 0
        scf = 0
        for f in fans:
            fb |= (1<<f)
            scf += cost[f]
        if fb.bit_count() <= K:
            if not any(m==fb for m,_ in c_list):
                c_list.append((fb, scf))
        bestc = 10**18; bm = None
        for m,sc in c_list:
            v = sc + 1
            if v < bestc:
                bestc = v; bm = m
        if bm is None:
            bm = fb; bestc = scf + 1
        cost[i] = bestc; bestM[i] = bm
        cuts[i] = sorted(c_list, key=lambda x: x[1])[:M]
    mapping = set()
    st = [name_to_idx[nm] for nm in outputs if nm in name_to_idx]
    vis = set()
    while st:
        u = st.pop()
        if u in vis: continue
        vis.add(u)
        bm = bestM[u]
        if bm is None: continue
        mapping.add(u)
        t = bm
        while t:
            lsb = t & -t; j = lsb.bit_length()-1; t ^= lsb
            if j in PI or topo_nodes[j].const_val is not None: continue
            if j not in vis: st.append(j)
    mapped = sorted(mapping)
    with open(solution_file, 'w') as f:
        f.write('.model ' + model + '\n')
        f.write('.inputs ' + ' '.join(inputs) + '\n')
        f.write('.outputs ' + ' '.join(outputs) + '\n')
        for i in mapped:
            nd = topo_nodes[i]
            bm = bestM[i]
            leaves = [j for j in range(N) if (bm>>j)&1]
            inputs_l = [idx_to_name[j] for j in leaves]
            k = len(inputs_l)
            Np = 1<<k
            full = (1<<Np) - 1
            masks = []
            for t in range(k):
                mm = 0
                for j in range(Np):
                    if (j>>t)&1: mm |= 1<<j
                masks.append(mm)
            val = {}
            for t,j in enumerate(leaves):
                val[j] = masks[t]
```

```
            vis2 = set(leaves)
            stk = [i]
            while stk:
                u = stk.pop()
                if u in vis2: continue
                vis2.add(u)
                for p in topo_nodes[u].fanins:
                    pj = name_to_idx.get(p.name)
                    if pj is not None and pj not in vis2:
                        stk.append(pj)
            cone = [j for j in topo if False]
            cone = [j for j in range(N) if j in vis2 and j not in leaves]
            for j in sorted(cone):
                v = topo_nodes[j]
                if v.const_val is not None:
                    val[j] = full if v.const_val==1 else 0
                else:
                    if v.patterns_zero:
                        zm = 0
                        for p in v.patterns_zero:
                            mm = full
                            for t,ch in enumerate(p):
                                fn = v.fanin_names[t]
                                pj = name_to_idx.get(fn)
                                if pj is None:
                                    mm = 0; break
                                vm = val[pj]
                                if ch=='1': mm &= vm
                                elif ch=='0': mm &= (~vm)&full
                            zm |= mm
                        val[j] = (~zm)&full
                    else:
                        om = 0
                        for p in v.patterns_one:
                            mm = full
                            for t,ch in enumerate(p):
                                fn = v.fanin_names[t]
                                pj = name_to_idx.get(fn)
                                if pj is None:
                                    mm = 0; break
                                vm = val[pj]
                                if ch=='1': mm &= vm
                                elif ch=='0': mm &= (~vm)&full
                            om |= mm
                        val[j] = om
            root = val[i]
            f.write('.names ' + ' '.join(inputs_l) + ' ' + nd.name + '\n')
            for j in range(Np):
                if (root>>j)&1:
                    pat = ''.join('1' if (j>>t)&1 else '0' for t in range(k))
                    f.write(pat + ' 1\n')
        f.write('.end\n')

```

In the fifth iteration, it introduces a dynamic pruning method by adapting the cut limit based on circuit size – more cuts for smaller circuits (higher quality), and fewer for large circuits (better yield).

```
# iteration 5: dynamic pruning of cut enumeration space

def solve(input_file: str, solution_file: str):
    from collections import deque
    raw = open(input_file).read().splitlines()
    lines = []
    i = 0
    while i < len(raw):
        l = raw[i].rstrip()
        if not l or l.lstrip().startswith('#'):
            i += 1
            continue
        if l.endswith('\\'):
            acc = l[:-1].rstrip()
            i += 1
            while i < len(raw):
                nl = raw[i].rstrip()
                if nl.endswith('\\'):
                    acc += ' ' + nl[:-1].rstrip()
                    i += 1
                else:
                    acc += ' ' + nl
```

```
                i += 1
                break
        lines.append(acc)
    else:
        lines.append(l)
        i += 1
class Node:
    __slots__ = ('name','fanin_names','patterns_one','patterns_zero','const_val','fanins','fanouts')
    def __init__(self,n):
        self.name = n
        self.fanin_names = []
        self.patterns_one = []
        self.patterns_zero = []
        self.const_val = None
        self.fanins = []
        self.fanouts = []
model = ''
inputs = []
outputs = []
nodes = {}
idx = 0
while idx < len(lines):
    l = lines[idx].strip()
    if not l:
        idx += 1; continue
    if l.startswith('.model'):
        parts = l.split()
        if len(parts) > 1: model = parts[1]
        idx += 1
    elif l.startswith('.inputs'):
        parts = l.split()
        for nm in parts[1:]:
            inputs.append(nm)
            if nm not in nodes: nodes[nm] = Node(nm)
        idx += 1
    elif l.startswith('.outputs'):
        parts = l.split()
        for nm in parts[1:]:
            outputs.append(nm)
        idx += 1
    elif l.startswith('.names'):
        parts = l.split()
        inps = parts[1:-1]; outp = parts[-1]
        if outp not in nodes: nodes[outp] = Node(outp)
        nd = nodes[outp]
        nd.fanin_names = inps[:]
        pats = []
        j = idx + 1
        while j < len(lines) and not lines[j].startswith('.'):
            s = lines[j].strip()
            if s: pats.append(s)
            j += 1
        if not inps:
            nd.const_val = 1 if pats and pats[0] == '1' else 0
        else:
            for pt in pats:
                sp = pt.split()
                if len(sp) == 1:
                    mask, bit = sp[0], '1'
                else:
                    mask, bit = sp[0], sp[1]
                if bit == '1': nd.patterns_one.append(mask)
                else: nd.patterns_zero.append(mask)
        idx = j
    elif l.startswith('.end'):
        break
    else:
        idx += 1
for nd in nodes.values():
    nd.fanins = [nodes[nm] for nm in nd.fanin_names if nm in nodes]
for nd in nodes.values():
    for f in nd.fanins:
        f.fanouts.append(nd)
reachable = set()
st = [nm for nm in outputs if nm in nodes]
while st:
    u = st.pop()
    if u in reachable: continue
    reachable.add(u)
    for p in nodes[u].fanin_names:
        if p in nodes and p not in reachable:
```

```
                st.append(p)
    nodes = {nm:nd for nm,nd in nodes.items() if nm in reachable}
    for nd in nodes.values():
        nd.fanins = [f for f in nd.fanins if f.name in nodes]
        nd.fanouts = [f for f in nd.fanouts if f.name in nodes]
    indeg = {nm: len(nd.fanins) for nm,nd in nodes.items()}
    dq = deque([nm for nm,d in indeg.items() if d == 0])
    topo = []
    while dq:
        u = dq.popleft(); topo.append(u)
        for w in nodes[u].fanouts:
            indeg[w.name] -= 1
            if indeg[w.name] == 0: dq.append(w.name)
    topo_nodes = [nodes[nm] for nm in topo]
    N = len(topo_nodes)
    name_to_idx = {nd.name: i for i, nd in enumerate(topo_nodes)}
    idx_to_name = [nd.name for nd in topo_nodes]
    PI = set(name_to_idx[nm] for nm in inputs if nm in name_to_idx)
    cost = [0] * N
    bestM = [None] * N
    K = 6
    if N <= 1500:
        M = 64
    elif N <= 3000:
        M = 48
    elif N <= 5000:
        M = 32
    else:
        M = 20
    cuts = [[] for _ in range(N)]
    def prune_cuts(items, limit):
        items_sorted = sorted(items, key=lambda x: (x[1], x[0].bit_count()))
        pr = []
        for m, sc in items_sorted:
            dom = False
            for pm, psc in pr:
                if psc <= sc and (pm & m) == pm:
                    dom = True
                    break
            if not dom:
                pr.append((m, sc))
                if len(pr) >= limit:
                    break
        return pr
    for i, nd in enumerate(topo_nodes):
        if i in PI or nd.const_val is not None:
            cost[i] = 0
            bestM[i] = None
            cuts[i] = [(1 << i, 0)]
            continue
        fans = [name_to_idx[nm] for nm in nd.fanin_names if nm in name_to_idx]
        fans.sort(key=lambda x: len(cuts[x]))
        c_list = None
        for f in fans:
            fcuts = cuts[f]
            if not fcuts:
                c_list = []
                break
            fcuts = fcuts[:M]
            if c_list is None:
                c_list = fcuts.copy()
            else:
                newm = {}
                for m1, sc1 in c_list:
                    for m2, sc2 in fcuts:
                        m = m1 | m2
                        if m.bit_count() <= K:
                            s2 = sc1 + sc2
                            prev = newm.get(m)
                            if prev is None or s2 < prev:
                                newm[m] = s2
                if not newm:
                    c_list = []
                    break
                c_list = prune_cuts(list(newm.items()), M)
        if not c_list:
            um = 0; usc = 0
            for f in fans:
                um |= (1 << f)
                usc += cost[f]
            c_list = [(um, usc)]
```

```
        um = 0; usc = 0
        for f in fans:
            um |= (1 << f)
            usc += cost[f]
        if um.bit_count() <= K and all(m != um for m, _ in c_list):
            c_list.append((um, usc))
        c_list = prune_cuts(c_list, M)
        bestc = 10**18; bm = None
        selfm = (1 << i)
        for m, sc in c_list:
            if m == selfm:
                continue
            v = sc + 1
            if v < bestc:
                bestc = v; bm = m
        if bm is None:
            bm = um
            bestc = usc + 1
        cost[i] = bestc
        bestM[i] = bm
        cuts[i] = c_list
    mapping = set()
    st = [name_to_idx[nm] for nm in outputs if nm in name_to_idx]
    vis = set()
    while st:
        u = st.pop()
        if u in vis: continue
        vis.add(u)
        bm = bestM[u]
        if bm is None: continue
        mapping.add(u)
        t = bm
        while t:
            lsb = t & -t
            j = lsb.bit_length() - 1
            t ^= lsb
            if j in PI or topo_nodes[j].const_val is not None: continue
            if j not in vis:
                st.append(j)
    mapped = sorted(mapping)
    with open(solution_file, 'w') as f:
        f.write('.model ' + model + '\n')
        f.write('.inputs ' + ' '.join(inputs) + '\n')
        f.write('.outputs ' + ' '.join(outputs) + '\n')
        for i in mapped:
            nd = topo_nodes[i]
            bm = bestM[i]
            leaves = [j for j in range(N) if (bm >> j) & 1]
            inputs_l = [idx_to_name[j] for j in leaves]
            k = len(inputs_l)
            Np = 1 << k
            full = (1 << Np) - 1
            masks = []
            for t in range(k):
                mm = 0
                for j in range(Np):
                    if (j >> t) & 1:
                        mm |= 1 << j
                masks.append(mm)
            val = {}
            for t, j in enumerate(leaves):
                val[j] = masks[t]
            seen = set(leaves)
            stk = [i]
            cone = []
            while stk:
                u = stk.pop()
                if u in seen: continue
                seen.add(u)
                cone.append(u)
                for p in topo_nodes[u].fanins:
                    pj = name_to_idx.get(p.name)
                    if pj is not None and pj not in seen:
                        stk.append(pj)
            cone.sort()
            for j in cone:
                v = topo_nodes[j]
                if v.const_val is not None:
                    val[j] = full if v.const_val == 1 else 0
                else:
                    if v.patterns_zero:
```

```
                        zm = 0
                        for p in v.patterns_zero:
                            mm = full
                            for t, ch in enumerate(p):
                                fn = v.fanin_names[t]
                                pj = name_to_idx.get(fn)
                                if pj is None:
                                    mm = 0
                                    break
                                vm = val[pj]
                                if ch == '1':
                                    mm &= vm
                                elif ch == '0':
                                    mm &= (~vm) & full
                            zm |= mm
                        val[j] = (~zm) & full
                    else:
                        om = 0
                        for p in v.patterns_one:
                            mm = full
                            for t, ch in enumerate(p):
                                fn = v.fanin_names[t]
                                pj = name_to_idx.get(fn)
                                if pj is None:
                                    mm = 0
                                    break
                                vm = val[pj]
                                if ch == '1':
                                    mm &= vm
                                elif ch == '0':
                                    mm &= (~vm) & full
                            om |= mm
                        val[j] = om
            root = val[i]
            f.write('.names ' + ' '.join(inputs_l) + ' ' + nd.name + '\n')
            for j in range(Np):
                if (root >> j) & 1:
                    pat = ''.join('1' if (j >> t) & 1 else '0' for t in range(k))
                    f.write(pat + ' 1\n')
        f.write('.end\n')
```

F Datasets

We summarize the original data sources for each problem in Table 23. All datasets are derived from real-world applications. We further partition or transform them into standardized input formats, ensuring the inclusion of both small-scale instances for demonstration purposes and large-scale instances for evaluation. For detailed data organization, please refer to our GitHub repository.

Table 23: Datasets used in our benchmark.

Problem	**Original Data Source**
Operator scheduling	EXPRESS [148]
Technology mapping	EPFL [6] and ISCAS85 [55]
Global routing	ISPD'24 Contest [80]
E-graph extraction	SmoothE [15]
Intra-op parallelism	ASPLOS'24 Contest [97]
Protein sequence design	Protein Data Bank (PDB) [37]
Mendelian error detection	Cost Function Library [123, 127]
Airline crew pairing	China Graduate Mathematical Modeling Competition'21 F [29]
Pickup and delivery w/ time windows	MetaPDPTW [77]