Proceedings of the Thirty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-26)
Special Track on AI4Tech: AI Enabling Critical Technologies

LLM-guided Cutting-plane Management for Mixed-integer Linear Programming

Zetao Zheng[1,2], **Zhe Wang**[1] and **Jie Shao**[1,2*]
[1]University of Electronic Science and Technology of China, Chengdu, China
[2]Sichuan Artificial Intelligence Research Institute, Yibin, China
ztzheng@uestc.edu.cn, wangzhe_cs@std.uestc.edu.cn, shaojie@uestc.edu.cn

Abstract

Cutting planes are central to mixed-integer linear programming (MILP) solving, yet their effectiveness hinges on expert tuning of separator configurations and hard-crafted cut-selection heuristics, creating a high barrier for non-specialists. Learning-based methods can reduce manual effort, but typically require large training datasets and often generalize poorly beyond the instance classes they are trained on. We propose an LLM-guided cutting-plane management framework that integrates large language models into the MILP solving pipeline. First, using chain-of-thought (CoT) prompting, the LLM infers an instance-specific separator configuration, deciding which separators to activate and how to set key parameters from the problem type and structural features. Then, it translates the evolving branch-and-bound state in natural language to perform stage-aware cut selection, choosing a high-quality subset of cuts that tightens the relaxation and improves overall solver performance. Leveraging the LLM's reasoning capabilities and rich background knowledge, our work removes dependence on domain-specific training data and substantially reduces reliance on expert-crafted configurations. Experiments show consistent improvements over SCIP's default settings, hard-crafted heuristics, and recent learning-based cut selection baselines.

1 Introduction

Mixed-integer linear programming (MILP) has been widely used in supply chain management [Andrade-Pineda *et al.*, 2017], production planning [Mahmoud *et al.*, 2023], scheduling [Chen, 2010], facility location [Caselli *et al.*, 2022] and bin packing [Mhiri, 2024]. A standard MILP can be formulated as follows:

$$z^* \triangleq \min_{\mathbf{x}} \{\mathbf{c}^\top \mathbf{x} | \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{R}^n, x_j \in \mathbb{Z} \text{ for all } j \in \mathcal{I}\}, \tag{1}$$

where $\mathbf{c} \in \mathbb{R}^n$ is the coefficient vector, $\mathbf{A} \in \mathbb{R}^{m \times n}$ is the constraint matrix, $\mathbf{b} \in \mathbb{R}^m$ is the constant vector, $\mathbf{x} \in \mathbb{R}^n$ is the vector of decision variables, $\mathcal{I} \subseteq \{1, \ldots, n\}$ denotes the set of indices of integer variables, and z^* denotes the optimal objective value of the problem. Modern MILP solvers typically adopt a branch-and-cut (B&C) framework, which combines a branch-and-bound (B&B) tree search with cutting-plane techniques. The B&B procedure systematically partitions the search space into smaller subproblems by branching on decision variables. At each node of the search tree, cutting planes are added to tighten the linear programming (LP) relaxation, thereby improving the lower bound and accelerating the solving process.

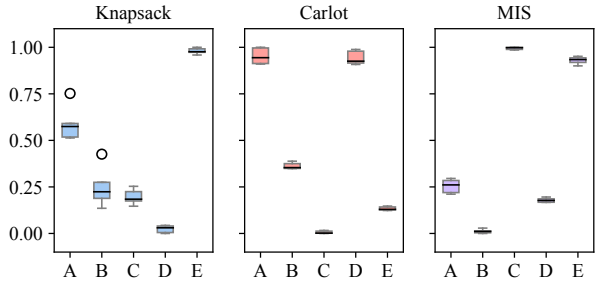


Figure 1: Comparison of separator configurations across three kinds of problems. The x-axis denotes five different separator configurations (A-E), while the y-axis shows the 0-1 normalized gap integral, which serves as a quality metric for solution performance (lower is better).

In cutting-plane-based MILP solving, solver performance is largely determined by cutting-plane management, which comprises two core components: separator configuration and cut selection. Modern MILP solvers implement various cutting-plane generating algorithms, also referred to as *separators*, to generate cutting planes that tighten the LP relaxations, such as *Gomory*, *clique* and *flowcover*. As illustrated in Figure 1, which reports the results under five different separator configurations (A–E), each configuration is executed five times using the solver's default parameter settings. From this figure, we observe high inter-type diversity in separator configurations: the performance varies substantially across different separator configurations for each problem type. For example, configuration D yields better results for both Carlot and MIS instances, but performs poorly on Knapsack instances. This indicates that different types of MILP problem

*Corresponding author: Jie Shao.

benefit from different separator settings, and that an inappropriate configuration can severely impair solution quality. However, separator configuration remains relatively underexplored in both the academic literature and solver implementations, where it is typically manually tuned by human experts. For cut selection, the effectiveness of a cut-filtering rule can vary substantially across different stages of the B&B search. However, most MILP solvers still rely on a single fixed, hand-crafted heuristic applied uniformly throughout the search, without adapting to the evolving tree state. Recent studies have explored using machine learning to improve cut selection [Gasse *et al.*, 2019; Huang *et al.*, 2022; Tang *et al.*, 2020; Paulus *et al.*, 2022]. However, these learning-based approaches typically require large amounts of training data and often generalize poorly to unseen instance distributions or problem types.

In this paper, we propose a novel cutting-plane management framework that incorporats LLMs to enhance MILP solving. First, we use CoT prompting to infer an instance-specific separator configuration, determining which separators to activate and how to set their key control parameters (e.g., separation frequency, the maximum number of separation rounds, and the maximum separation depth) based on the problem type and structural characteristics (e.g., numbers of variables and constraints, integrality ratio, and objective structure). Next, since the most effective cuts can vary across different stages of the branch-and-bound search, we condition on the evolving tree state and use the LLM to generate stage-aware cut-selection heuristics and executable code to filter the candidate cut pool, thereby improving overall solver performance. We integrate our approach into the state-of-the-art open-source MILP solver SCIP [Bestuzheva *et al.*, 2023] and show that LLM-guided cut management yields notable improvements in solution quality compared with competitive heuristic-based and learning-based baselines.

In summary, our contributions are as follows:

- We introduce an LLM-guided cutting-plane management framework for MILP solving that automates both separator configuration and cut selection, substantially reducing reliance on manual tuning and expert-crafted rules.
- We develop (i) an instance-structure-aware separator tuning mechanism that uses CoT reasoning to jointly decide separator activation and key parameter values, and (ii) a stage-aware heuristic evolution mechanism that monitors the evolving B&B state and generates executable cut-selection policies (code) to filter cuts. Together, these mechanisms significantly improve the effectiveness of cutting-plane-based MILP solving.
- We conduct comprehensive experiments on five widely studied optimization problems, demonstrating that our LLM-guided approach achieves competitive performance.

2 Related Work

2.1 Machine Learning for MILP

In recent years, leveraging machine learning to enhance the performance of MILP solvers gains considerable attention [Bengio *et al.*, 2021; Gasse *et al.*, 2021; Gasse *et al.*, 2019]. Traditional solvers rely heavily on hand-crafted heuristics to make critical decisions during the solving process [Achterberg, 2007], and recent studies explore replacing these heuristics with learned models [Bengio *et al.*, 2021]. This paradigm shift leads to notable improvements in various solver components, including cut selection [Tang *et al.*, 2020; Paulus *et al.*, 2022; Khalil *et al.*, 2016], variable branching [Zarpellon *et al.*, 2021; Balcan *et al.*, 2018; Gasse *et al.*, 2019], node prioritization [He *et al.*, 2014; Sabharwal *et al.*, 2012], column generation [Yuan *et al.*, 2024], and primal heuristic selection [Khalil *et al.*, 2017; Hendel *et al.*, 2018]. In this work, we focus on cut selection management, a key factor influencing solver performance.

In the context of cut selection, a number of learning-based approaches emerge to prioritize cuts by learning a scoring function that reflects their effectiveness [Huang *et al.*, 2022; Tang *et al.*, 2020; Paulus *et al.*, 2022]. For instance, Tang *et al.* [2020] introduce a reinforcement learning method that learns to evaluate Gomory cuts [Edmunds and Bard, 1992] and selects the one with the highest predicted score. Similarly, Paulus *et al.* [2022] propose a lookahead-based rule that selects cuts expected to yield the greatest improvement in the dual bound, and learns this rule via imitation learning. Extending beyond the selection of a single best cut, Huang *et al.* [2022] formulate the task as a multiple instance learning problem, where a scoring function identifies a fixed proportion of high-quality cuts. In addition to determining which cuts to keep, Wang *et al.* [2023] introduce hierarchical sequential model (HEM) to investigate the preferred order in which selected cuts should be applied—an aspect that also affects the efficiency of solving MILPs. However, most existing methods overlook the importance of deciding the appropriate number and sequence of cuts. Meanwhile, Turner *et al.* [2023] explore a hybrid approach that learns to combine expert-designed scoring rules via learned weights. On the theoretical front, Balcan *et al.* [2021] provide provable guarantees for learning-based cut selection strategies. While cut selection has received increasing attention, the complementary task of separator configuration, which involves deciding both which cut generators to activate and how to set their key parameters, remains relatively underexplored. A recent study by Li *et al.* [2023] proposes a data-driven framework for learning separator activation policies, demonstrating that informed separator selection can reduce the cut candidate space and improve solver efficiency.

2.2 LLM for Optimization

Recent research demonstrates significant progress in leveraging LLMs to automate and enhance various aspects of optimization. For instance, Huang *et al.* [2025] propose a customizable framework for training LLMs specifically tailored for optimization modeling in the context of operations research (OR). Beyond training domain-specific models, many studies focus on using general-purpose LLMs to support optimization workflows. For example, Liu *et al.* [2024] and Shi *et al.* [2025] employ LLMs to generate heuristics that guide evolutionary algorithms such as genetic algorithms (GAs), demonstrating the effectiveness of LLMs in heuristic design

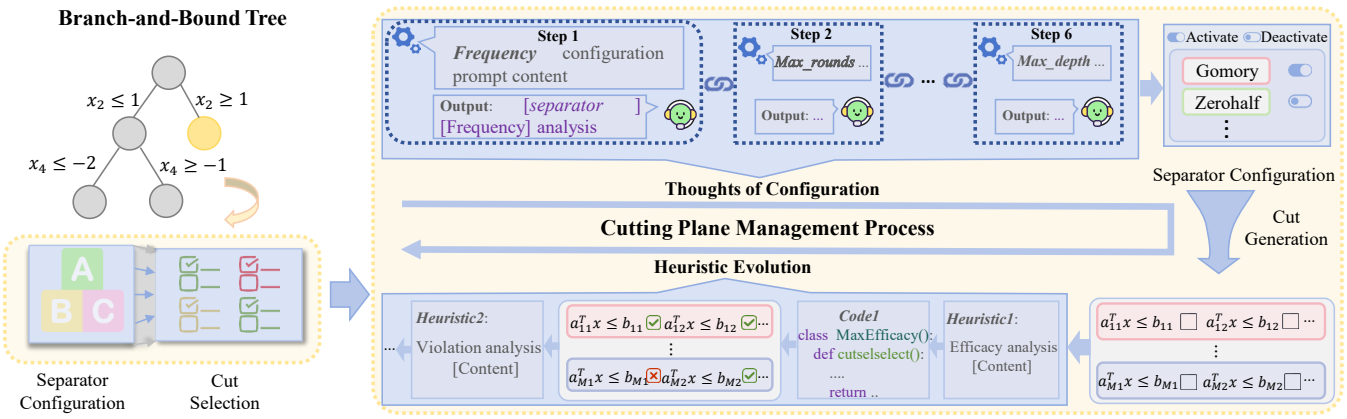


Figure 2: Overview of our approach.

and evolution. Similarly, Jiang *et al.* [2025a] introduce an LLM-based framework that integrates network topology and domain knowledge to address combinatorial optimization (CO) problems. Zhang and Luo [2025] further push the boundary by developing reasoning-based LLM agents that automate the entire OR workflow by interpreting natural language problem descriptions, translating them into code, and solving the resulting optimization problems. In addition, Lawless *et al.* [2025] incorporate LLMs into MILP optimization to mitigate the cold-start problem by leveraging textual descriptions of both problem instances and available separators. More recently, LLMs are being more tightly coupled with the solver loop through performance-driven search over heuristics or executable code. LLM-LNS [Ye *et al.*, 2025] uses a dual-layer self-evolutionary agent to generate neighborhood-selection strategies for MILP large neighborhood search. LLM4Solver [Jiang *et al.*, 2025b] explores program-space evolution, using LLMs to generate and refine diving-heuristic code. Related heuristic-evolution frameworks, such as Hercules [Wu *et al.*, 2025] and Monte Carlo tree search for automated heuristic design (MCTS-AHD) [Wu *et al.*, 2025; Zheng *et al.*, 2025], further improve LLM-generated heuristics by incorporating task-aware abstractions and lowering evaluation cost, for example via fitness prediction.

Despite these advances, most prior work emphasizes domain-specific LLM training or generates external heuristics and code, while largely overlooking how LLMs can enhance the core exact components of mainstream solvers, such as cutting-plane methods. In this work, we leverage LLMs' mathematical reasoning to guide both separator configuration and adaptive cut selection, improving solution quality.

3 Problem Formulation

Modern MILP solvers commonly adopt a branch-and-cut framework, which combines branch-and-bound with cutting planes to iteratively tighten LP relaxations and prune the search space. Within this framework, separator configuration specifies which separations are activated to generate candidate cuts, while cut selection decides which cuts are added to the relaxation at each separation round. We formally define three components below to facilitate a clearer understanding of our approach.

Definition 1 (Branch-and-Cut)**.** *Modern MILP solvers adopt the B&C paradigm, which integrates cutting-plane techniques into the B&B framework [Mitchell, 2002]. The solver builds a search tree by recursively solving LP relaxations and branching on fractional variables, as Figure 2 displays. To strengthen these relaxations, especially at the root node, cutting planes are added to tighten the feasible region, improving bounds and reducing tree size [Bengio* et al.*, 2021].*

Definition 2 (Separator Configuration)**.** *Given an MILP solver with M separator algorithms, configuring separator is not only about deciding which separators to enable, but also about setting the separator-specific parameters that control their aggressiveness and computational overhead. We formalize a separator configuration as $\mathcal{F}_m = (z_m, \boldsymbol{\theta}_m)_{m=1}^{M}$, where $z_m \in \{0, 1\}$ indicates whether separator F_m is activated, and $\boldsymbol{\theta}_m$ denotes its parameter vector (e.g., separation frequency, the maximum number of separation rounds, and the maximum separation depth). The difficulty lies in exploring both the combinatorial space of separator activations and the coupled parameter space for the activated separators. Even considering z_m alone yields 2^M combinations, which is already huge for solvers such as SCIP and Gurobi with $M = 17$ and $M = 21$ separators, respectively. Parameter tuning of $\boldsymbol{\theta}_m$ further enlarges the search space and is highly instance-dependent, typically requiring substantial expert experience. LLMs offer a promising alternative by inferring reasonable activation decisions and parameter settings directly from semantic instance descriptions and embedded priors.*

Definition 3 (Cutting Plane Selection)**.** *Given the MILP problem in Eq.* (1)*, we drop all its integer constraints to obtain its LP relaxation, which takes the form of:*

$$z_{LP}^{*} \triangleq \min_{\mathbf{x}} \{\mathbf{c}^{\top}\mathbf{x} | \mathbf{A}\mathbf{x} \le \mathbf{b}, \mathbf{x} \in \mathbb{R}^{n}\}. \tag{2}$$

Given the LP relaxation in Eq. (2)*, cutting planes are linear inequalities introduced to tighten the relaxation without excluding any integer-feasible solutions of the original MILP. These cuts are generated by a set of activated separators and*

applied iteratively during the solving process. The cutting-plane procedure typically includes: (i) solving the current LP relaxation, (ii) generating a pool of candidate cuts $\mathcal{C}$, (iii) selecting a high-quality subset $\mathcal{S} \subseteq \mathcal{C}$, and (iv) adding $\mathcal{S}$ to the LP model to produce an updated relaxation. Crucially, the best cut-selection rule is not static: it should adapt to the evolving search stage and tree state, where the trade-off between bound improvement and LP overhead can change markedly. Hence, effective and adaptive cut selection is a key driver of MILP solver performance [Tang et al.*, 2020].*

4 Methodology

Different separators are designed to exploit specific structural properties of MILP instances, and their effectiveness often depends on proper parameter settings. For instances with diverse scales and objective structures, choosing and tuning an appropriate separator configuration is crucial, as it directly shapes the candidate cut pool and, ultimately, solver performance. Once candidate cuts are generated, cut selection is equally important and should adapt to the evolving B&B tree to balance relaxation tightening against computational overhead. Unlike prior approaches that rely on expert-crafted rules or large training datasets, we propose to leverage LLMs' semantic understanding of instance characteristics and solver states to automatically configure separators and produce stage-aware cut-selection strategies.

4.1 Separator Configuration via CoT

Modern MILP solvers (e.g., SCIP) provide a rich portfolio of separator algorithms, each designed to exploit particular geometric or combinatorial structures in the LP relaxation. However, solver performance is highly sensitive not only to which separators are enabled, but also to how their key parameters are set (e.g., invocation frequency, round limits, cut limits, and depth restrictions). Tuning these choices typically requires optimization expertise and extensive trial-and-error. Moreover, indiscriminately activating all separators in the solver and setting their key parameters to large values can significantly increase computational overhead and, in turn, make the overall solving process more difficult. To address this, we propose an LLM-guided separator configuration method that leverages the mathematical priors and reasoning capabilities of LLMs to produce an instance-specific configuration via CoT reasoning, jointly deciding separator activation and separator-level hyperparameters.

Formally, let the available separator set be $\mathbb{F} = \{F_1, \ldots, F_M\}$. Our goal is to infer a separator configuration $\mathcal{F} = \{(z_m, \boldsymbol{\theta}_m)\}_{m=1}^{M}$ that will be used throughout the solving process, where $z_m \in \{0, 1\}$ indicates whether F_m is activated, and $\boldsymbol{\theta}_m$ collects its key parameters. Given an instance $\mathcal{P}$, the LLM infers $\mathcal{F}$ through CoT prompting:

$$\mathcal{F} = f_{\text{LLM}}\big(\texttt{Prompt}(\mathcal{P}, \mathbb{F})\big). \tag{3}$$

Concretely, the prompt encodes a compact structural profile of $\mathcal{P}$, including: (i) problem type (e.g., Set Covering, Knapsack), capturing high-level combinatorial structure; (ii) instance scale (numbers of variables and constraints); (iii) variable composition (e.g., integrality ratio and domain statistics); and (iv) objective/constraint structure (e.g., coefficient sparsity, magnitude ranges, sign patterns and density). The prompt also enumerates the available separator set $\mathbb{F}$ and the admissible parameter ranges for each F_m, forcing the LLM to choose only supported algorithms and valid parameters.

$$\mathcal{F}_m = \begin{cases} z_m = f_{\text{LLM}}^{(0)}(\mathcal{P}, F_m)\,, \\ \theta_m^{(j)} = f_{\text{LLM}}^{(j)}\Big(\mathcal{P}, F_m, z_m, \theta_m^{(1:j-1)}\Big)\,. \end{cases} \tag{4}$$

Our CoT prompting explicitly decomposes separator configuration into an ordered, step-by-step reasoning procedure. For each separator F_m, as formalized in Eq. (4), the LLM first determines a binary activation decision z_m. Conditioned on the activated separator z_m, it then infers the separator parameters sequentially, where j indexes the j-th parameter and $\theta_m^{(1:j-1)}$ denotes the set of previously inferred parameters. In the current implementation, the LLM reasons about six core parameters. These six parameters include the invocation frequency, the maximum number of separation rounds at the root and non-root nodes, the maximum number of cuts added per round at the root and non-root nodes, and the maximum B&B depth at which the separator is allowed to operate. Importantly, this reasoning pipeline is fully extensible: if additional separator parameters are exposed (e.g., caps on the number of cuts added and other separator-specific controls), they can be incorporated by appending extra reasoning steps to the chain, without changing the overall framework.

4.2 Cut Selection via Heuristic Evolution

Given a pool of candidate cuts $\mathcal{C}$ generated by the activated separators, a cut-selection heuristic defines a rule for filtering $\mathcal{C}$ to select a subset of cuts to be added to the LP relaxation. As B&B progresses, the solver moves across different search regimes (e.g., root vs. deep nodes, easy vs. hard subtrees), so a single fixed cut-selection heuristic is often suboptimal.

In this work, we introduce a heuristic evolution mechanism driven by an LLM to adapt cut selection to the current B&B stage. Let s_t denote the B&B tree state at cut-selection call t, summarizing the current position and progress of the search tree (e.g., depth, node count, gap, LP iterations). Let h_t denote the cut-selection heuristic used at time t, and let π_t denote the observed short-horizon performance signal after applying it (e.g., gap decrease, LP time, node growth). We first use the LLM to interpret the evolving state-performance stream $(s_\tau, \pi_\tau)_{\tau \le t}$ in natural language and assess whether the current heuristic h_t is effective under s_t. If not, the LLM proposes a more suitable heuristic description (a heuristic thought) and the corresponding executable cut-selection policy (code), yielding an updated heuristic h_{t+1}:

$$h_{t+1} = \mathcal{G}_{\text{LLM}}\Big(s_t; (s_\tau, \pi_\tau)_{\tau \le t}, ; h_t\Big), \tag{5}$$

where $\mathcal{G}_{\text{LLM}}$ denotes the LLM-driven generator that, conditioned on the current tree state, produces a better cut-selection heuristic h_{t+1} for the next search stage.

Because generating high-quality heuristics (and code) via the LLM can be time-consuming, prior work often relies on expensive evolutionary procedures such as genetic algorithms (GA) to iteratively score and evolve candidates [Ye *et*

	Set Covering (x = 1000, y = 500)			Maximum Independent Set (x = 500, y = 1953)			Multiple Knapsack (x = 720, y = 72)			MIK (x = 413, y = 346)			Corlat (x = 466, y = 486)		
	PDI↓	PD gap↓	Primal bound↓	PDI ↓	PD gap↓	Primal bound ↓	PDI↓	PD gap ↓	Primal bound ↓	PDI↓	PD gap ↓	Primal bound ↓	PDI↓	PD gap ↓	Primal bound↓
Default	75.49	0.0	248.70	218.24	0.0	218.69	18.29	0.0	400.50	285.30	0.0	-4516.60	4246.40	0.88	nan
Random	246.78	0.0	241.79	617.95	0.0	224.29	13.14	0.0	428.77	1019.40	0.0	-4516.60	2544.26	0.05	nan
NV	132.85	0.0	243.25	499.55	0.0	226.01	12.16	0.0	434.07	294.76	0.0	-4516.60	4031.79	0.08	nan
Eff	134.90	0.0	241.63	215.29	0.0	218.69	14.18	0.0	428.37	764.30	0.0	-4516.60	1066.24	0.09	nan
HEM	**44.92**	0.0	228.17	**37.45**	0.0	226.71	19.36	0.0	428.77	798.52	0.02	nan	2014.43	0.13	nan
Ours (Nocut)	58.62	0.0	228.17	40.70	0.0	226.25	**4.12**	0.0	435.72	525.79	0.06	nan	1637.12	0.09	nan
Ours (Nosep)	148.66	0.0	248.70	286.40	0.0	**218.69**	43.02	0.0	**400.00**	515.42	0.0	-4516.60	12386.59	0.24	nan
Ours (All)	58.17	0.0	**228.17**	58.19	0.0	226.25	14.46	0.0	435.72	**283.76**	0.0	**-4516.60**	**1404.27**	**0.01**	**265.24**
p-value	3.2e-4	1.8e-4	6.8e-4	3.1e-4	4.8e-4	9.1e-4	6.0e-4	6.1e-4	7.6e-4	2.3e-4	1.5e-4	3.0e-4	2.1e-4	7.6e-4	-

Table 1: Performance comparison with other models on the five datasets. Let y denote the average number of constraints and x denote the average number of variables. 'nan' denotes failure to obtain an optimal solution.

al., 2025; Wu *et al.*, 2025]. In this work, we build an offline strategy repository $\mathcal{D} = (u_k, h_k)_{k=1}^{K}$, where each entry stores a heuristic h_k together with a concise semantic summary u_k (the heuristic thought) describing its intended operating regime and best-use conditions. At runtime, given the current state s_t, the LLM matches s_t to the most semantically compatible thought u_k and selects the corresponding heuristic:

$$h_t \leftarrow h_{k^\star}, \qquad k^\star = \arg \max_{k \in \{1,\dots,K\}} \mathrm{Sim}_{\mathrm{LLM}}(s_t, ; u_k). \quad (6)$$

Here, $k^\star$ denotes the index of the strategy in the repository $\mathcal{D} = \{(u_k, h_k)\}_{k=1}^{K}$ whose heuristic description u_k is considered by the LLM to be the most semantically compatible with the current B&B state s_t. We then activate the corresponding cut-selection heuristic $h_{k^\star}$. This realizes heuristic evolution as stage-aware switching among a curated set of effective cut-selection behaviors, rather than costly GA-style population evolution.

5 Experiments

We conduct a series of experiments to address the following research questions: (1) **Overall Performance**: How does our method compare with heuristic-based, learning-based, and LLM-based approaches in solution quality across benchmark datasets, particularly on large-scale instances? (2) **Efficiency**: How does the time consumption of our method compare with the baselines? (3) **Ablation Study**: How do the separator configuration and cut selection modules individually contribute to the overall performance? (4) **Interpretability**: What can we observe from visualizing the cuts selected by the LLM?

5.1 Datasets

We evaluate our approach on five NP-hard MILP benchmarks. Three of them—Set Covering [Balas and Ho, 2009; Ye *et al.*, 2025], Maximum Independent Set (MIS) [Bergman *et al.*, 2016], and Knapsack [Scavuzzo *et al.*, 2022]—are synthetic instances generated following the methodology of Gasse *et al.* [2019]. Specifically, the Set Covering (small) dataset contains 1,000 variables and 500 constraints, while the (large) dataset contains 200,000 variables and 200,000 constraints. the Maximum Independent Set dataset has 500 variables and 1,593 constraints; and the Knapsack dataset comprises 720 variables and 72 constraints. The remaining two benchmarks, MIK [Atamtürk, 2003] and CORLAT [Gomes *et al.*, 2008], are real-world problem instances commonly used for evaluating MILP solvers. The MIK dataset includes 413 variables and 346 constraints, while the CORLAT dataset contains 466 variables and 486 constraints. A summary of dataset statistics, including the number of variables (x) and the number of constraints (y), is provided in Table 1.

5.2 Setup

We use SCIP 8.0 [Bestuzheva *et al.*, 2023] as the backend solver for all experiments. As a state-of-the-art open-source solver, SCIP is widely adopted in machine learning research for combinatorial optimization [Gasse *et al.*, 2019; Huang *et al.*, 2025]. After extensive evaluation of various LLMs, we adopt Qwen3-30B or Claude-3.5 to guide both separator configuration and cut selection, depending on the problem characteristics. All other SCIP parameters are kept at their default settings to ensure fair and reproducible comparisons. A time limit of 300 seconds is imposed for solving each instance. More implementation details can be found in our code repository and online appendix: https://github.com/BeeeeenWANG/LLM-guided-cutting-plane-management.

5.3 Baselines

The first baseline uses the built-in cut selection strategy of SCIP 8.0, referred to as Default. The second baseline selects a fixed number of cuts at random and is denoted as Random. The third and fourth baselines adopt the normalized violation (NV) and efficacy (Eff) heuristics, respectively, to guide cut selection. The fifth baseline is a state-of-the-art learning-based method, hierarchical sequential model (HEM) [Wang *et al.*, 2023], which learns cut selection policies through hierarchical reinforcement learning. The sixth baseline is an LLM-based solver, LLM-LNS [Ye *et al.*, 2025], which uses LLM-generated heuristics to guide large neighborhood search for MILPs. Additionally, we evaluate two ablation variants of our method. NoSep disables the LLM-guided separator configuration and uses SCIP's default separators and parameters, while retaining the LLM-guided cut selection. NoCut disables the LLM-guided cut selection and relies on SCIP's default cut selection rule, while preserving the LLM-guided separator configuration.

5.4 Metrics

We adopt three widely used evaluation metrics: average primal-dual gap integral (PDI), primal bound, primal-dual

Model	Set Covering (small) (x = 1000, y = 500)		Set Covering (large) (x = 200000, y = 200000)	
	Primal Integral↓	Primal bound↓	Primal Integral↓	Primal bound↓
Default	**6.78**	**211.43**	1.25e+07	17890.35
Random	22.5	248.91	4.61e+07	20140.72
NV	9.75	236.21	1.92e+07	18430.68
Eff	7.92	221.67	1.43e+07	18020.95
HEM	8.94	229.34	9.76e+07	17380.52
LLM-LNS	13.21	228.17	3.32e+07	16742.43
Ours	6.95	220.14	**1.03e+07**	**15475.31**

Table 2: Performance comparison with baselines (including LLM-based method) on small and large-scale datasets.

Model	Set Covering		Maximum Independent Set		MIK	
	PDI↓	PD gap↓	PDI↓	PD gap↓	PDI↓	PD gap↓
Qwen3-30B	195.78	0.0	**58.19**	0.0	283.76	0.0
DeepSeek-V3	215.97	0.0	82.85	0.0	282.26	0.0
Kimi-K2	220.96	0.0	83.63	0.0	385.54	0.0
GPT-4o	233.90	0.0	84.43	0.0	359.00	0.0
Claude-3.5	**58.17**	0.0	77.76	0.0	419.88	0.0
Gemini-2.5	185.55	0.0	69.62	0.0	**270.70**	0.0
Grok-3	185.55	0.0	106.53	0.0	437.32	0.0

Table 3: Performance of different LLMs on the three datasets.

gap (PD gap) and primal integral. PD integral measures the area under the primal-dual gap curve over time, reflecting both the convergence speed and solution quality throughout the solving process. A lower PD integral indicates faster and more stable convergence. Primal bound refers to the best feasible objective value found during the solving process. For minimization problems (which is standard in MILP), a lower primal bound implies a better solution. Primal-dual gap (PD gap) is the relative difference between the primal and dual bounds at termination. It quantifies how close the solution is to optimality. A lower PD gap indicates better optimality and is therefore preferred. A PD gap of 0 means that the primal and dual bounds have converged, indicating that a provably optimal solution has been found. In addition to these metrics, we also compare the solving time, where shorter time reflects better efficiency. Primal integral measures the area under the primal-bound trajectory over time (relative to the optimal or best-known objective value), reflecting the solver's anytime performance; a lower primal integral indicates that high-quality feasible solutions are found earlier and improved more steadily.

5.5 Results

Performance Comparison (Answering Q.1)

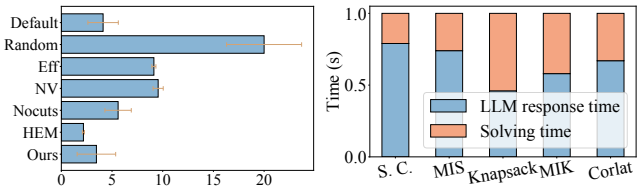
The comparison with other baselines is presented in Table 1, where the best-performing model is shown in bold, and the strongest baseline in each case is underlined. Our LLM-guided cut management approach and its two ablated variants consistently demonstrate competitive performance across all benchmarks. Specifically, on relatively easier datasets such as Set Covering and Maximum Independent Set, the learning-based method HEM performs strongly in terms of solution quality. However, as the problem complexity increases—such as in more challenging datasets with more intricate cut structures—HEM struggles to maintain its performance, while our method begins to show clear advantages. It is also noteworthy that applying LLM-guided separator configuration alone can sometimes achieve strong results without the additional cost of cut selection. This is especially evident on datasets like Knapsack, where the number of generated cuts is small or relatively simple. In such cases, introducing cut selection may add overhead without significantly improving solution quality. The p-values also confirm that the experimental results are of statistical significance.

Since LLM-LNS is not built on a tree-search solving mechanism, it does not maintain valid dual bounds. To enable a fair comparison with LLM-based solvers, Table 2 reports results on Set Covering at both small and large scales using primal integral and primal bound. On the small instance, Default and our method achieve the lowest primal integrals, indicating strong anytime performance, while our method further improves the final primal bound compared with hand-crafted rules and learning-based baselines. Random performs worst, confirming the importance of informed cut management. NV and Eff provide moderate improvements but remain inferior to adaptive strategies. On the large-scale instance, our method attains the best overall performance, with the lowest primal integral and the strongest primal bound among all methods, demonstrating superior scalability and robustness. HEM and LLM-LNS achieve competitive final bounds but suffer from significantly larger primal integrals, reflecting weaker convergence behavior over time. Overall, the results show that the proposed LLM-guided cutting-plane management effectively balances convergence speed and solution quality across problem scales.

We further evaluate the impact of different LLMs as reasoning engines, including Qwen3-30B, DeepSeek-V3, Kimi-K2-Instruct, GPT-4o, Claude-3.5, Gemini-2.5, and Grok-3. A brief comparison is presented in Table 3, with detailed results provided in our online appendix. All models are able to find optimal solutions (i.e., PD gap equals 0) on the three evaluated datasets. The primary differences lie in their performance in the PDI (primal-dual integral) metric, which reflects the LLMs' inference efficiency and reasoning effectiveness. Among these models, Qwen3-30B, Claude-3.5, and Gemini-2.5 achieve the best overall results, exhibiting lower PDI values. In particular, Qwen3-30B and Claude-3.5 deliver strong performance while maintaining relatively low inference costs. While models such as Grok-3 and Gemini-2.5 demonstrate competitive efficiency, they occasionally yield slightly less stable cut quality. Based on this analysis, we select Qwen3-30B and Claude-3.5 as our primary reasoning engines, enabling a flexible trade-off between solution quality and inference budget.

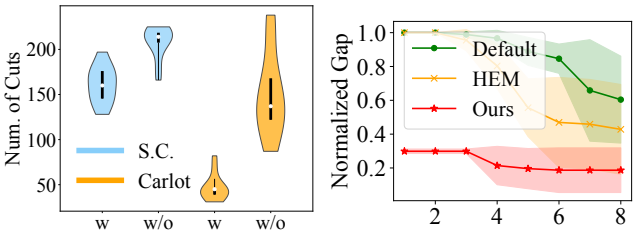
Time Consuming Analysis (Answering Q.2)

We report the average runtime of all methods on the Set Covering dataset in Figure 3(a), averaged over 10 trials. For fairness, we exclude the LLM inference time for separator configuration, since it is a one-time preprocessing step performed before optimization, whereas all methods report only the solving time during cut selection. Under this protocol, our approach does not exhibit a clear advantage in terms of runtime compared with other methods. HEM approaches achieve the lowest reported solving time, although their training cost is not included. In contrast, random cut selection exhibits the



(a) Set Covering (b) Proportion of solving time

Figure 3: Time consumption analysis.



(a) Impact of separator configuration on two datasets (b) Impact of cut selection on the Set Covering dataset

Figure 4: Ablation study on LLM-guided separator configuration and cut selection.

largest variance due to the inconsistent quality of the selected cuts.

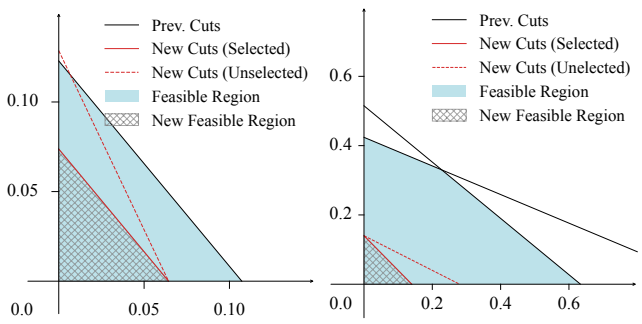
To understand the runtime bottleneck, Figure 3(b) breaks down the time spent on LLM reasoning. Across most datasets, LLM inference accounts for nearly half of the total solving time. While it helps select high-quality cuts, its cost remains high, motivating future work on accelerating LLM inference for more efficient cut selection.

Ablation Study on Separator Configuration (Answering Q.3)

We analyze the number of cuts generated by separators during the cut selection process on the Set Covering and Carlot datasets. To evaluate the impact of the separator configuration module, we compare our approach with (denoted as 'w') and without (denoted as 'w/o') this component. The results in Figure 4(a) show that, on both datasets, particularly Carlot, the number of cuts considered for selection is significantly reduced when the separator configuration is applied. This demonstrates that separator configuration effectively narrows the search space by filtering out unproductive separators, which plays a critical role in improving cut selection efficiency.

Ablation Study on Cut Selection (Answering Q.3)

We analyze the LLM-guided cut selection by comparing it with two typical baselines in terms of primal gap: the default setting of the SCIP solver and HEM. The results in Figure 4(b) show that our LLM-guided approach achieves faster convergence from a lower initial gap and exhibits greater stability in reducing the gap, with less fluctuation than the other two methods. These advantages can be attributed to the LLM's inherent mathematical reasoning capabilities, which allow it to select effective cuts rather than relying solely on simple heuristics or learned parameters.



(a) Set Covering (b) Multiple Knapsack

Figure 5: Visualization of cut selection.

Visualization on Cut Selection (Answering Q.4)

During the cutting plane procedure, the solver typically relaxes integer constraints using linear programming and iteratively adds cutting constraints to eliminate non-integer solutions. This process progressively tightens the feasible region. Figures 5(a) and (b) illustrate how the feasible region is incrementally reduced through successive cuts. In these figures, black solid lines represent the cuts selected in the previous round, and the blue-shaded area enclosed by these lines and the coordinate axes denotes the current feasible region under linear relaxation. Red lines indicate candidate cuts: the red solid line is the one selected by the LLM for the next round, while the red dashed lines are those rejected by the LLM.

After the red solid cut is added, the feasible region (gray-shaded area) becomes smaller, enabling more non-integer solutions to be excluded and bringing the relaxed region closer to the true integer feasible set. This visual evidence demonstrates the effectiveness of selecting appropriate separators and adaptive cut-selection heuristics.

6 Conclusion and Future Work

In this paper, we propose an LLM-guided cutting-plane management framework for MILP that integrates LLM reasoning into both separator configuration and cut selection. By leveraging chain-of-thought reasoning, the LLM automatically infers instance-specific separator activation and parameter settings, while a stage-aware heuristic switching mechanism adaptively selects effective cut-selection strategies based on the evolving B&B state. Experimental results on diverse benchmarks show that the proposed method consistently outperforms baselines in solution quality, demonstrating the potential of LLMs for enhancing core components of exact MILP solvers.

Looking ahead, several directions worth further exploration. First, we plan to accelerate the reasoning process of LLMs to enable faster solving. Furthermore, integrating fine-tuned, domain-specific LLMs may provide a better trade-off between inference cost and decision quality, paving the way for scalable and efficient LLM-guided optimization pipelines.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 62276047), China Postdoctoral Science Foundation (No. 2025M771590 and No. GZC20251104), and Yibin Science and Technology Program (No. 2024JC007 and No. 2025SF004).

References

[Achterberg, 2007] Tobias Achterberg. *Constraint integer programming*. PhD thesis, Berlin Institute of Technology, 2007.

[Andrade-Pineda *et al.*, 2017] José Luis Andrade-Pineda, David Canca, and Pedro L. Gonzalez-R. On modelling non-linear quantity discounts in a supplier selection problem by mixed linear integer optimization. *Ann. Oper. Res.*, 258(2):301–346, 2017.

[Atamtürk, 2003] Alper Atamtürk. On the facets of the mixed-integer knapsack polyhedron. *Math. Program.*, 98(1-3):145–175, 2003.

[Balas and Ho, 2009] Egon Balas and Andrew Ho. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. In *Combinatorial optimization*, pages 37–60. 2009.

[Balcan *et al.*, 2018] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, pages 353–362, 2018.

[Balcan *et al.*, 2021] Maria-Florina Balcan, Siddharth Prasad, Tuomas Sandholm, and Ellen Vitercik. Sample complexity of tree search configuration: Cutting planes and beyond. In *Advances in Neural Information Processing Systems, NeurIPS 2021*, pages 4015–4027, 2021.

[Bengio *et al.*, 2021] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *Eur. J. Oper. Res.*, 290(2):405–421, 2021.

[Bergman *et al.*, 2016] David Bergman, André A. Ciré, Willem-Jan van Hoeve, and John N. Hooker. *Decision Diagrams for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. 2016.

[Bestuzheva *et al.*, 2023] Ksenia Bestuzheva, Mathieu Besançon, Weikun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros M. Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco E. Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, et al. Enabling research through the SCIP optimization suite 8.0. *ACM Trans. Math. Softw.*, 49(2):22:1–22:21, 2023.

[Caselli *et al.*, 2022] Giulia Caselli, Giomaria Columbu, Manuel Iori, and Carlo Alberto Magni. Mixed integer linear programming for CO2 emissions minimization in a waste transfer facility location problem. In *Proceedings of the 10th International Network Optimization Conference, INOC 2022*, pages 1–6, 2022.

[Chen, 2010] Zhi-Long Chen. Integrated production and outbound distribution scheduling: Review and extensions. *Oper. Res.*, 58(1):130–148, 2010.

[Edmunds and Bard, 1992] Thomas A. Edmunds and Jonathan F. Bard. An algorithm for the mixed-integer nonlinear bilevel programming problem. *Ann. Oper. Res.*, 34:149–162, 1992.

[Gasse *et al.*, 2019] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems, NeurIPS 2019*, pages 15554–15566, 2019.

[Gasse *et al.*, 2021] Maxime Gasse, Simon Bowly, Quentin Cappart, Jonas Charfreitag, Laurent Charlin, Didier Chételat, Antonia Chmiela, Justin Dumouchelle, Ambros M. Gleixner, Aleksandr M. Kazachkov, Elias B. Khalil, et al. The machine learning for combinatorial optimization competition (ML4CO): results and insights. In *NeurIPS 2021 Competitions and Demonstrations Track, 6-14 December 2021*, pages 220–231, 2021.

[Gomes *et al.*, 2008] Carla P. Gomes, Willem Jan van Hoeve, and Ashish Sabharwal. Connections in networks: A hybrid approach. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference, CPAIOR 2008*, pages 303–307, 2008.

[He *et al.*, 2014] He He, Hal Daumé III, and Jason Eisner. Learning to search in branch and bound algorithms. In *Advances in Neural Information Processing Systems, NIPS 2014*, pages 3293–3301, 2014.

[Hendel *et al.*, 2018] Gregor Hendel, Matthias Miltenberger, and Jakob Witzig. Adaptive algorithmic behavior for solving mixed integer programs using bandit algorithms. In *Operations Research Proceedings 2018*, pages 513–519, 2018.

[Huang *et al.*, 2022] Zeren Huang, Kerong Wang, Furui Liu, Hui-Ling Zhen, Weinan Zhang, Mingxuan Yuan, Jianye Hao, Yong Yu, and Jun Wang. Learning to select cuts for efficient mixed-integer programming. *Pattern Recognit.*, 123:108353, 2022.

[Huang *et al.*, 2025] Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. ORLM: A customizable framework in training large models for automated optimization modeling. *Oper. Res.*, 73(6):2986–3009, 2025.

[Jiang *et al.*, 2025a] Shuo Jiang, Min Xie, and Jianxi Luo. Large language models for combinatorial optimization of design structure matrix. *Proceedings of the Design Society*, 5:2201–2210, 2025.

[Jiang *et al.*, 2025b] Xia Jiang, Yaoxin Wu, Minshuo Li, Zhiguang Cao, and Yingqian Zhang. Large language

models as end-to-end combinatorial optimization solvers. In *Advances in Neural Information Processing Systems, NeurIPS 2025*, 2025.

[Khalil *et al.*, 2016] Elias Boutros Khalil, Pierre Le Bodic, Le Song, George L. Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the Thirtieth Conference on Artificial Intelligence, AAAI 2016*, pages 724–731, 2016.

[Khalil *et al.*, 2017] Elias B. Khalil, Bistra Dilkina, George L. Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, pages 659–666, 2017.

[Lawless *et al.*, 2025] Connor Lawless, Yingxi Li, Anders Wikum, Madeleine Udell, and Ellen Vitercik. Llms for cold-start cutting plane separator configuration. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research, CPAIOR 2025*, pages 51–69, 2025.

[Li *et al.*, 2023] Sirui Li, Wenbin Ouyang, Max B. Paulus, and Cathy Wu. Learning to configure separators in branch-and-cut. In *Advances in Neural Information Processing Systems 36, NeurIPS 2023*, 2023.

[Liu *et al.*, 2024] 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 *Forty-first International Conference on Machine Learning, ICML 2024*, 2024.

[Mahmoud *et al.*, 2023] Ayman Mahmoud, Mohammed Hichame Benbitour, Zied Jemaï, Evren Sahin, and Marc Baratte. Integrated shipment and production planning: A mixed-integer linear programming approach with multiple suppliers and due dates. In *IEEE International Conference on Networking, Sensing and Control, ICNSC 2023*, pages 1–6, 2023.

[Mhiri, 2024] Mariem Mhiri. A mixed-integer programming model for the bin packing problem with piecewise linear loading cost and time windows. In *IEEE International Conference on Industrial Engineering and Engineering Management, IEEM 2024*, pages 395–399, 2024.

[Mitchell, 2002] John E. Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. In *Handbook of Applied Optimization*, pages 65–77. 2002.

[Paulus *et al.*, 2022] Max B. Paulus, Giulia Zarpellon, Andreas Krause, Laurent Charlin, and Chris J. Maddison. Learning to cut by looking ahead: Cutting plane selection via imitation learning. In *International Conference on Machine Learning, ICML 2022*, pages 17584–17600, 2022.

[Sabharwal *et al.*, 2012] Ashish Sabharwal, Horst Samulowitz, and Chandra Reddy. Guiding combinatorial optimization with UCT. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems CPAIOR 2012*, pages 356–361, 2012.

[Scavuzzo *et al.*, 2022] Lara Scavuzzo, Feng Yang Chen, Didier Chételat, Maxime Gasse, Andrea Lodi, Neil Yorke-Smith, and Karen I. Aardal. Learning to branch with tree mdps. In *Advances in Neural Information Processing Systems, NeurIPS 2022*, 2022.

[Shi *et al.*, 2025] Yiding Shi, Jianan Zhou, Wen Song, Jieyi Bi, Yaoxin Wu, and Jie Zhang. Generalizable heuristic generation through large language models with meta-optimization. *CoRR*, abs/2505.20881, 2025.

[Tang *et al.*, 2020] Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*, pages 9367–9376, 2020.

[Turner *et al.*, 2023] Mark Turner, Thorsten Koch, Felipe Serrano, and Michael Winkler. Adaptive cut selection in mixed-integer linear programming. *Open J. Math. Optim.*, 4:1–28, 2023.

[Wang *et al.*, 2023] Zhihai Wang, Xijun Li, Jie Wang, Yufei Kuang, Mingxuan Yuan, Jia Zeng, Yongdong Zhang, and Feng Wu. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. In *The Eleventh International Conference on Learning Representations, ICLR 2023*, 2023.

[Wu *et al.*, 2025] Xuan Wu, Di Wang, Chunguo Wu, Lijie Wen, Chunyan Miao, Yubin Xiao, and You Zhou. Efficient heuristics generation for solving combinatorial optimization problems using large language models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2025*, pages 3228–3239, 2025.

[Ye *et al.*, 2025] Huigen Ye, Hua Xu, An Yan, and Yaoyang Cheng. Large language model-driven large neighborhood search for large-scale MILP problems. In *Forty-second International Conference on Machine Learning, ICML 2025*, 2025.

[Yuan *et al.*, 2024] Haofeng Yuan, Lichang Fang, and Shiji Song. A reinforcement-learning-based multiple-column selection strategy for column generation. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024*, pages 8209–8216, 2024.

[Zarpellon *et al.*, 2021] Giulia Zarpellon, Jason Jo, Andrea Lodi, and Yoshua Bengio. Parameterizing branch-and-bound search trees to learn branching policies. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*, pages 3931–3939, 2021.

[Zhang and Luo, 2025] Bowen Zhang and Pengcheng Luo. Or-llm-agent: Automating modeling and solving of operations research optimization problem with reasoning large language model. *CoRR*, abs/2503.10009, 2025.

[Zheng *et al.*, 2025] Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in llm-based automatic heuristic design. In *Forty-second International Conference on Machine Learning, ICML 2025*, 2025.