FROM TUNING TRIALS TO MULTI-LEVEL EXPERIENCE: LLM-GUIDED MIP SOLVER CONFIGURATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixed-integer programming (MIP) solvers play a vital role in solving large-scale combinatorial optimization problems, which often originate from important real-world applications. It is well-known that the parameter configuration of the MIP solvers has a significant impact on the solving efficiency, while users often rely on static defaults or a single dataset-wide configuration for all instances. However, different instances often favor substantially different configurations, as real-world MIP instances usually originate from diverse sources and they are highly heterogeneous. To tackle this problem, we propose **H**ierarchical **E**xperience **G**raph for LLM-Guided Parameter **Config**uration (**HEG-Config**), which uses an LLM to generate an instance-specific configuration by exploiting tuning experience accumulated across instances. The key challenges to develop HEG-Config are twofold: (1) LLMs lack specialized tuning experience, and (2) tuning itself must balance generalization and instance-specific adaptation. By introducing the novel **hierarchical experience graph**, which can effectively distill historical tuning trials into multi-level experience from instance-specific rules to general tuning strategies, HEG-Config can well address the aforementioned challenges simultaneously. Then, during inference, by retrieving relevant experience at different levels, HEG-Config enables the LLM to integrate general and instance-specific tuning experience to generate effective configurations. Experiments demonstrate that HEG-Config achieves better aggregate performance than the baselines, consistently improving solving efficiency over the default configuration and reducing the primal-dual integral within the given time limit by 21%–31%.

1 INTRODUCTION

Mixed-integer programming (MIP) solvers are powerful tools for solving large-scale combinatorial optimization problems. In recent years, MIP solvers have been successfully applied to a wide range of real-world applications, such as production planning, logistics, and scheduling. Modern solvers such as SCIP (Achterberg, 2009) and Gurobi (Gurobi Optimization, LLC, 2026) expose thousands of discrete, conditionally dependent parameters that have a substantial impact on solving efficiency (Li et al., 2025). In practice, users often rely on static default settings for simplicity. Even when traditional algorithm configuration (AC) methods (Ye et al., 2022; Lindauer et al., 2022; Franceschi et al., 2025) are used, they typically tune a single configuration over a set of training instances, assuming that these instances are drawn from the same homogeneous distribution (Iommazzo et al., 2022; Schede et al., 2022). However, this assumption is often violated in practice: real-world MIP instances usually originate from diverse sources and they are highly heterogeneous (Xu et al., 2010), making it difficult for a single configuration to perform well overall and leaving the one-configuration-for-all paradigm inherently limited.

Existing instance-specific methods that tailor a configuration to each instance (Georges et al., 2018; Hosny & Reda, 2023; Song et al., 2023) partially address this issue. Such methods search over possible configurations and then, for a given instance, either learn a preference over candidate configurations and select one, or map the instance through a model to a configuration. However, such methods operate only within the set of configurations seen or available during search. A target instance may require a combination of parameter values that no single explored configuration realizes, even though each of these values is supported by tuning results from some instance. For such an instance, these methods can at best pick the closest among the configurations explored, rather than

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

adapt individual parameters to its structure. Moreover, these methods depend on components fitted to the training data, which are sensitive to distribution shift. As a result, in practice these methods are typically developed and validated on mixtures of relatively homogeneous instance classes, and they struggle to scale to genuinely heterogeneous benchmarks or to generalize to unseen instance distributions.

In this paper, we propose **H**ierarchical **E**xperience **G**raph for LLM-Guided Parameter **Config**uration (**HEG-Config**), a novel framework that leverages an LLM to generate parameter configurations for MIP solvers. Rather than selecting from a fixed set of configurations explored during tuning, HEG-Config directly generates parameter assignments for a given instance, allowing individual parameters to be adapted to its structure. The key challenges in developing HEG-Config are twofold. First, although LLMs reason and generate well, they do not inherently possess expertise in MIP solver configuration and parameter interaction. Second, generality and instance-specificity pull against each other: configuration knowledge that is too general may overlook the characteristics of a particular instance, whereas knowledge tied too closely to specific instances may not transfer effectively to new ones. To address these challenges, HEG-Config distills historical tuning trajectories into experience cards and progressively extends them into a hierarchical experience graph, where lower levels preserve instance-specific rules and higher levels summarize more general strategies. During inference, a hierarchy-aware retrieval mechanism provides this multi-level experience to the LLM, enabling it to integrate general and instance-specific experience and synthesize an effective configuration for the given instance.

To demonstrate HEG-Config's ability to generalize across heterogeneous and unseen instance distributions, we evaluate it on MIPLIB 2017 (Gleixner et al., 2021), a heterogeneous benchmark of real-world MIP instances, and further validate it via cross-dataset evaluation on D-MIPLIB (Huang et al., 2024). Experiments demonstrate that HEG-Config achieves better aggregate performance than the baselines, consistently improving solving efficiency over the default configuration and reducing the primal-dual integral within the given time limit by 21%–31%, while remaining effective across diverse datasets without domain-specific customization. Moreover, we evaluate HEG-Config under the conventional setup where all methods are trained and evaluated on a mixture of relatively homogeneous datasets, and HEG-Config still attains the best performance.

2 Preliminaries

2.1 Mixed-Integer Programming and SCIP Solver

Mixed-integer programming (MIP) is a class of optimization problems in which a subset of the decision variables is restricted to integer values, providing a general framework for modeling a wide range of combinatorial optimization problems. Solving general MIP problems is NP-hard in the worst case, making them computationally challenging. Modern MIP solvers combine branch-and-bound with techniques such as cutting planes, presolve, and primal heuristics to progressively improve primal and dual bounds and reduce the optimality gap. When the gap is closed, the solver can certify the optimality of the incumbent solution.

SCIP (Achterberg, 2009) is a widely used branch-cut-and-price framework for MIP, integrating branch-and-bound with cutting planes, LP-based relaxations, primal heuristics, and other solver components. Its solving performance can be highly sensitive to parameter settings, and different configurations may lead to substantially different search behaviors and solving times for the same MIP instance. SCIP provides more than 2,500 configurable parameters, spanning branching rules, cutting planes, primal heuristics, and other aspects of the solving process. Moreover, many parameters exhibit conditional dependencies, resulting in a highly structured and complex parameter space. In this paper, we focus on generating instance-specific parameter configurations for SCIP to improve its performance on individual MIP instances. Further details of the SCIP parameter space are provided in Appendix B.1.

2.2 Evaluation Metrics for MIP Solvers

MIP solvers are typically run under a finite time limit. For instances that are solved to optimality within this limit, the runtime is a suitable measure of solving efficiency. Many instances, however, cannot be solved to optimality in time. For these, the relative optimality gap is commonly used,

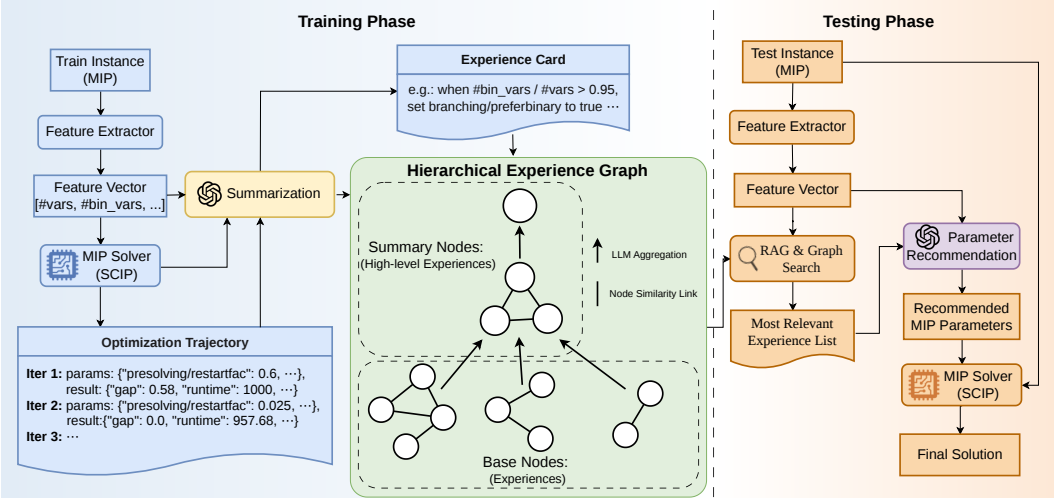


Figure 1: **Overview of HEG-Config.** (1) Training phase: tuning trajectories are converted into feature-conditioned experience cards, then aggregated bottom-up to form a hierarchical graph with high-level strategy summaries. (2) Testing phase: for a new instance, relevant cards are retrieved and expanded along the hierarchy to obtain multi-granularity context for configuration generation.

defined as the relative difference between the best feasible solution (primal bound) and the dual bound, following the SCIP definition (Achterberg, 2009). The gap nevertheless has two limitations: it carries no information for instances that are solved to optimality, whose gap is zero regardless of how fast the solver was, and for the remaining instances it only reflects the terminal state and ignores the solving process over time. The primal-dual integral (PDI) addresses both by accumulating the relative gap over time:

$$\text{PDI} = \int_0^T \text{gap}(t)\, dt$$

where $\text{gap}(t)$ denotes the relative gap at time t, thereby serving as a holistic measure of overall solving efficiency. In this paper, we use the PDI as the optimization objective during the offline construction phase and the NPDI, which normalizes the PDI against the default configuration, as the main evaluation metric. More details about these metrics are provided in Appendix B.2.

3 Hierarchical Experience Graph for LLM-Guided Parameter Configuration

We propose HEG-Config, a framework that leverages LLMs to generate parameter configurations for MIP solvers by distilling historical tuning trajectories into a hierarchical experience graph. Specifically, HEG-Config consists of two parts: **(i) offline experience distillation** (Training phase), which first distills historical tuning trajectories into experience cards (Section 3.1) and then constructs a hierarchical experience graph from these cards (Section 3.2); and **(ii) online hierarchy-aware retrieval** (Testing phase), which retrieves relevant experience at multiple granularities to guide configuration generation (Section 3.3). A visual overview is shown in Figure 1.

3.1 Experience Distillation

We first distill historical tuning trajectories into experience cards, which serve as the basic units of the experience graph. Each training MIP instance x_i is represented by a 147-dimensional feature vector $\mathbf{f}_i$, extracted using the official feature extractor from MIPLIB 2017 (Gleixner et al., 2021); these features characterize the structural properties of the instance and provide the context for distilling transferable configuration experience. For each training instance, we obtain a tuning trajectory using the Tree-structured Parzen Estimator (TPE), a Bayesian optimization method well suited to large discrete parameter spaces such as SCIP's. TPE iteratively proposes a configuration θ_t and

records its solving results, such as the gap, runtime, and primal-dual integral, producing a trajectory:

$$\mathcal{T}_i = \{(\theta_t, \mathbf{y}_t)\}_{t=1}^{T}, \tag{1}$$

where $\mathbf{y}_t$ denotes the solving results of configuration θ_t. We run $T = 50$ trials per instance.

Given the instance features $\mathbf{f}_i$ and the tuning trajectory $\mathcal{T}_i$, the LLM distills $\mathcal{T}_i$ into an experience card c_i under a strict template (see Appendix E). Each card contains three components: a set of **quantitative rules**, an experience summary, and an error analysis. The quantitative rules form the core, whereas the remaining components provide supporting explanations. Each quantitative rule condenses the observed tuning outcomes into a feature-conditioned relationship between the instance structure and parameter selection, thereby transforming instance-specific observations into reusable configuration rule:

$$\phi(\mathbf{f}) \ \rightarrow \ \theta, \tag{2}$$

where $\phi(\mathbf{f})$ specifies a condition on the instance features (e.g., "bin_vars $>$ 1000 and constr_matrix_density < 0.005") and θ specifies a recommended setting for a subset of the parameters (e.g., setting `branching/preferbinary` to `true`).

3.2 Hierarchical Graph Construction

The experience cards obtained in Section 3.1 capture the configuration experience of individual instances. To derive more general configuration experience, we construct a hierarchical experience graph $\mathcal{G}$ from these cards. Specifically, the experience cards form the base level V_0 of the graph $\mathcal{G} = (V, E)$, where $V = \bigcup_{k\geq 0} V_k$ and $E = \bigcup_{k\geq 0} E_k$. We then progressively introduce higher-level summary nodes, where each node at level $k > 0$ summarizes a group of related nodes at level $k-1$. Thus, lower levels preserve more specific experiences, while higher levels capture increasingly general configuration strategies.

To construct the hierarchical experience graph, we first store the base-level experience cards in a retrieval-augmented generation (RAG) backend, and then establish connectivity among them with a topology function $\Phi : V_0 \times V_0 \rightarrow \mathbb{R}_{\geq 0}$. By assigning a relatedness weight to every pair of cards, Φ defines the edge set E_0 of the base-level graph. The specific implementation of Φ can be instantiated with different retrieval backends, such as vector similarity in naive RAG (Lewis et al., 2020) or structural connectivity in LightRAG (Guo et al., 2024).

The hierarchy is then built bottom-up through local aggregation. At each level, we identify sufficiently large local neighborhoods of unsummarized nodes and compress each neighborhood into a higher-level summary node. For a node v, let $L(v) = \{u \in N(v) : \text{level}(u) = \text{level}(v) \wedge \neg\text{summarized}(u)\}$ denote its active neighbors, define the candidate neighborhood as $C(v) = L(v) \cup \{v\}$, and let τ denote the aggregation threshold. When $|C(v)| \geq \tau$, we create a summary node p by condensing the contents of the nodes in $C(v)$:

$$p = \text{Summarize}\big(\{\text{content}(u) : u \in C(v)\}\big), \quad \text{level}(p) = \text{level}(v) + 1, \tag{3}$$

and assign p as the parent of every node in $C(v)$. The edges between p and other nodes at this level are then induced from the connectivity of their children: for two higher-level nodes $p, q \in V_{k+1}$, their edge weight w_{pq} is given, for $\Omega \neq \emptyset$, by

$$w_{pq} = \frac{1}{|\Omega|} \sum_{(u,z)\in\Omega} w(u, z), \qquad \Omega = \{(u, z) \mid u \in \text{child}(p),\ z \in \text{child}(q),\ (u, z) \in E_k\}; \tag{4}$$

otherwise, no edge is created between p and q. Once assigned, these nodes are marked as summarized and excluded from subsequent aggregation at the same level.

This bottom-up process is repeated recursively: newly created summary nodes become the candidates for aggregation at the next level, allowing increasingly general configuration strategies to emerge. The resulting hierarchy connects concrete tuning cases with increasingly abstract strategies, providing multiple levels of experience for hierarchy-aware retrieval. The complete construction procedure is described in Algorithm 1 and Algorithm 2.

3.3 HIERARCHY-AWARE EXPERIENCE RETRIEVAL

Given a new instance with features $\mathbf{f}$, HEG-Config first retrieves relevant experience cards at the base level and then expands the retrieved experiences along the hierarchy to incorporate increasingly general configuration strategies. The retrieval procedure consists of three steps:

1. **Base-level retrieval:** Using the new instance's feature description as the retrieval query, the RAG backend retrieves a set of relevant cards $\mathcal{R}_0$ from the base level V_0.
2. **Upward propagation:** For each seed $v \in \mathcal{R}_0$, HEG-Config ascends from v to its parent and continues upward, retrieving summary nodes at increasingly higher levels. These summary nodes abstract common patterns shared by multiple lower-level experiences and provide progressively more general configuration strategies.
3. **Neighbor expansion:** At each reached level k, for every visited node p, HEG-Config additionally retrieves its Top-$\Psi(k)$ neighboring nodes ranked by edge weight w_{pq}, where $\Psi(k)$ is a non-increasing function of k.[1] The expanded neighbors are added to the context but do not themselves trigger further upward propagation.

Let $\mathcal{R}_k$ denote the set of nodes collected at level k, consisting of the summary nodes reached by upward propagation at level k together with their horizontally expanded neighbors. The retrieved nodes are aggregated into a multi-level context $\mathcal{R}_{\text{all}} = \bigcup_k \mathcal{R}_k$, which combines concrete quantitative rules from base-level experience cards with increasingly general strategy summaries from higher levels. The LLM then generates the final solver configuration conditioned on the instance features and the retrieved experience:

$$\theta = \text{LLM}\big(\mathbf{f}, \mathcal{R}_{\text{all}}\big). \tag{5}$$

The complete retrieval procedure is detailed in Algorithm 3.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTINGS

Parameter Selection. SCIP exposes a large parameter space, so we focus on a representative subset of 25 key parameters selected based on their influence on the solution path and their functional roles. These parameters cover the core components of the branch-cut-and-price framework, namely presolving, cut generation, branching, node selection, primal heuristics, and LP solving. The complete list of selected parameters is provided in Appendix C.1.

Dataset and Task Partitioning. As our main benchmark, we use MIPLIB 2017 (Gleixner et al., 2021), a highly heterogeneous dataset covering diverse problem families and scales. We pre-solve the instances using SCIP under the default configuration and retain the 175 for which a feasible solution is found within a 600s limit, splitting them into 125 for constructing the offline hierarchical experience graph and 50 for evaluation. To further validate our method, we additionally evaluate on multiple sub-datasets from D-MIPLIB (Huang et al., 2024) as cross-dataset test sets. For further analysis, we use MIPLIB and four D-MIPLIB datasets (CA-easy, SC-easy, CFLP-medium, and MIRP-medium), which cover different difficulty levels and problem domains. Details about the datasets are provided in Appendix C.4.

Evaluation Protocol. We evaluate all methods under a 1000s time limit. Performance is primarily measured by the normalized primal-dual integral (NPDI), computed at the dataset level as $\text{NPDI} = \sum_i \text{PDI}_i \,/\, \sum_i \text{PDI}_i^{\text{default}}$, where PDI_i is the primal-dual integral of instance i under a given configuration and $\text{PDI}_i^{\text{default}}$ that under the default configuration. Lower values indicate higher solving efficiency, with values below 1 denoting an improvement over the default. We additionally report the relative optimality gap as a secondary metric in our analyses. For a dataset, the reported gap is the sum of the gaps over all its instances, where instances for which no feasible solution is found are assigned a gap of $10{,}000$. Detailed definitions of these metrics are provided in Appendix B.2.

[1] Higher-level nodes provide more general strategy summaries, so fewer neighbors are needed to avoid redundant context.

Table 1: **NPDI for different methods**. Lower values indicate better performance, with values below 1 indicating an improvement over the default. We mark **the best values** in bold and underline the second-best values.

	Hydra						MILPTune	KNN	HEG-Config
	rf	dt	lsvc	lr	knn	svc			
Main benchmark (MIPLIB)									
MIPLIB	1.10	1.17	1.11	1.05	1.08	1.07	0.91	0.93	**0.79**
Cross-dataset evaluation (D-MIPLIB)									
# Easy Dataset									
CA	1.96	1.17	**0.43**	**0.43**	**0.43**	0.45	0.56	1.14	0.72
CFLP	0.97	1.23	0.96	0.97	1.21	1.22	0.74	0.82	**0.43**
GISP	**0.44**	0.91	0.81	2.29	2.13	1.67	1.02	0.84	0.75
OTS	1.49	1.46	3.28	3.27	1.44	0.97	1.45	0.90	**0.86**
SC	0.98	0.31	**0.30**	0.32	0.97	0.92	0.50	0.85	0.69
Mean	1.17	1.02	1.16	1.46	1.24	1.05	0.85	0.91	**0.69**
# Medium Dataset									
CA	2.22	1.01	0.76	0.77	0.77	0.82	**0.66**	0.99	0.81
CFLP	1.16	1.82	1.55	1.56	1.22	1.12	0.64	0.70	**0.36**
GISP	**0.32**	1.20	0.94	1.98	2.35	1.21	1.10	0.81	0.95
MIRP	0.98	1.00	0.95	0.91	0.94	1.02	1.28	1.02	**0.82**
OTS	2.06	1.00	2.33	2.31	1.78	1.00	1.89	1.30	**0.83**
SC	0.99	0.50	**0.40**	**0.40**	0.99	0.98	0.90	1.24	0.85
Mean	1.29	1.09	1.16	1.32	1.34	1.03	1.08	1.01	**0.77**

Baselines. We compare HEG-Config against three instance-specific configuration methods: (i) **Hydra** (Georges et al., 2018), a feature-based algorithm selection method with six classifier variants (rf, dt, lsvc, lr, knn, svc), which trains pairwise comparators over a portfolio of candidate configurations to produce a full ranking; (ii) **MILPTune** (Hosny & Reda, 2023), which trains a graph-convolutional instance encoder and performs KNN in the learned embedding space; and (iii) **KNN**, which retrieves the configuration of the nearest neighbor using the MIPLIB 2017 feature extractor (Gleixner et al., 2021). Moreover, the default configuration is an implicit baseline, since its NPDI is always 1. For more details of baselines, please refer to Appendix C.3.

Implementation Details. We use `qwen3-30b-a3b-instruct-2507` (Team, 2025) as the core model. The prompt templates are provided in Appendix E. For the RAG backend, we use LightRAG (Guo et al., 2024) with `qwen-text-embedding-v4` (Zhang et al., 2025). The topology function Φ then defines the base graph's edge weights by aggregating the relationships among the cards in LightRAG's knowledge graph. The aggregation threshold τ is set to 5. Given the current training scale, the constructed graph consists of 4 levels (L0–L3); thus, we set the neighbor expansion count to decrease linearly with the level k as $\Psi(k) = 4 - k$. Experiments are primarily conducted on CPU platforms using SCIP 10.0.0 (Hojny et al., 2025), where the tuning trajectories are sampled in parallel with 50 processes.

4.2 Main Results

Table 1 reports the NPDI of each method, where lower values indicate higher solving efficiency and values below 1 denote an improvement over the default configuration, whose NPDI is always 1. In addition to the NPDI, more detailed results for each dataset are provided in Table 10 of Appendix G.1.

HEG-Config delivers the strongest overall performance. HEG-Config achieves the best aggregate among all compared methods NPDI on all three groups – MIPLIB, the Easy D-MIPLIB group, and the Medium D-MIPLIB group – with values of 0.79, 0.69, and 0.77 (i.e., efficiency gains of roughly 21%, 31%, and 23% over the default). We interpret this advantage in terms of how HEG-Config uses historical experience: rather than selecting among previously discovered configurations,

Table 2: **Ablation of hierarchy and neighbor expansion in HEG-Config**. The last row reports the default SCIP configuration for reference.

	MIPLIB		CA-easy		SC-easy		CFLP-medium		MIRP-medium	
	Gap↓	NPDI↓	Gap↓	NPDI↓	Gap↓	NPDI↓	Gap↓	NPDI↓	Gap↓	NPDI↓
HEG-Config (full)	16.60	0.79	0.00	0.72	0.00	0.69	0.00	0.36	19.53	0.82
w/o Hierarchical	3964.85	0.94	0.00	0.90	0.00	0.83	0.01	0.77	10020.03	1.04
w/o Expand	36.87	0.91	0.00	0.89	0.00	0.81	0.01	0.54	10019.93	1.05
Default	4404.62	1.00	0.00	1.00	0.00	1.00	0.01	1.00	10022.90	1.00

Table 3: **Ablation of prompt type**. The last row reports the default SCIP configuration for reference. The bold rows denote **the settings adopted in our main experiments**.

	MIPLIB		CA-easy		SC-easy		CFLP-medium		MIRP-medium	
	Gap↓	NPDI↓	Gap↓	NPDI↓	Gap↓	NPDI↓	Gap↓	NPDI↓	Gap↓	NPDI↓
Flexible	4374.22	1.02	0.00	1.26	0.00	0.81	0.01	0.63	20026.47	1.07
Strict	**16.60**	**0.79**	**0.00**	**0.72**	**0.00**	**0.69**	**0.00**	**0.36**	**19.53**	**0.82**
Default	4404.62	1.00	0.00	1.00	0.00	1.00	0.01	1.00	10022.90	1.00

HEG-Config generates an instance-specific configuration by combining feature-conditioned rules retrieved at multiple granularities from the distilled experience graph. This distinction matters because conventional configuration methods either select from existing configurations (as in Hydra), or match a single similar instance and adopt its best configuration (as in KNN and MILPTune). They are therefore confined to configurations already observed during tuning and can only pick one as an atomic whole, without being able to adjust individual parameters on demand. HEG-Config, in contrast, recombines parameter-level experience in ways that no single historical configuration captures.

HEG-Config generalizes robustly across heterogeneous datasets and problem settings. Beyond aggregate performance, HEG-Config retains its advantage when the problem family and instance difficulty change: in the cross-dataset evaluation, it keeps the NPDI below 1 on every reported dataset, whereas competing methods deteriorate substantially on individual datasets, with several Hydra variants exceeding an NPDI of 2 and MILPTune or KNN occasionally falling above 1. This robustness is consistent with the abstraction introduced by the hierarchical experience: retrieval is not tied to a single similar instance but combines concrete quantitative rules with higher-level strategies shared across a broader group of instances, so the retrieved experience remains useful even when direct instance-level similarity to the new instance is limited.

HEG-Config still performs best under the conventional homogeneous-mix setup. In our introduction, we argue that existing methods are typically developed and validated on mixtures of a few relatively homogeneous instance classes. To verify that HEG-Config remains competitive in this conventional setup rather than only on the heterogeneous benchmark, we additionally train all methods on a mixture of the CA-easy, SC-easy, CFLP-medium, and MIRP-medium datasets and evaluate them on these same datasets. Table 11 reports the results, and further details are provided in Appendix G.2. HEG-Config again attains the lowest average NPDI in this setup, confirming that its advantage holds under the homogeneous-mix training scenario as well.

4.3 Ablation

We evaluate the contributions of the hierarchy and neighbor expansion through two ablation variants: `w/o Hierarchical`, which removes the hierarchy and retains only the flat set of base-level cards, and `w/o Expand`, which keeps the hierarchy but disables neighbor expansion during retrieval. We further investigate two key design choices: the prompt type and the aggregation threshold τ. In addition to the NPDI, we report the relative gap, which offers a more fine-grained view of the performance differences.

Both the hierarchy and neighbor expansion are essential to HEG-Config, with the hierarchical experience playing a more prominent role. As reported in Table 2, both variants degrade

Table 4: **Ablation of aggregation threshold τ**. The last row reports the default SCIP configuration for reference. The bold rows denote **the settings adopted in our main experiments**.

	MIPLIB		CA-easy		SC-easy		CFLP-medium		MIRP-medium	
	Gap↓	NPDI↓	Gap↓	NPDI↓	Gap↓	NPDI↓	Gap↓	NPDI↓	Gap↓	NPDI↓
2	19.86	0.86	0.00	1.02	0.00	0.77	0.01	0.62	25.38	0.87
5	**16.60**	**0.79**	**0.00**	**0.72**	**0.00**	**0.69**	**0.00**	**0.36**	**19.53**	**0.82**
10	62.39	0.92	0.00	1.15	0.00	0.77	0.02	0.45	10025.74	1.00
Default	4404.62	1.00	0.00	1.00	0.00	1.00	0.01	1.00	10022.90	1.00

Table 5: **Impact of hierarchy levels in HEG-Config**. The last row reports the default SCIP configuration for reference.

	MIPLIB		CA-easy		SC-easy		CFLP-medium		MIRP-medium	
	Gap↓	NPDI↓	Gap↓	NPDI↓	Gap↓	NPDI↓	Gap↓	NPDI↓	Gap↓	NPDI↓
Full (L0–L3)	16.60	0.79	0.00	0.72	0.00	0.69	0.00	0.36	19.53	0.82
L0 only	3964.85	0.94	0.00	0.90	0.00	0.83	0.01	0.77	10020.03	1.04
L0–L1	67.21	0.93	0.00	0.86	0.00	0.82	0.01	0.56	23.26	1.08
L1–L3	52.15	0.93	0.00	0.87	0.00	0.82	0.00	0.46	10020.25	1.05
L3 only	3854.53	0.95	0.00	0.93	0.00	0.82	0.01	0.82	10045.18	1.05
Default	4404.62	1.00	0.00	1.00	0.00	1.00	0.01	1.00	10022.90	1.00

performance on all five datasets, with the NPDI rising above the full model in every case. Taking MIRP-medium as an example, both ablation variants exceed an NPDI of 1, and each fails to find a feasible solution for one instance[2]. The two nonetheless contribute unequally: removing the hierarchy inflates the MIPLIB gap sharply, far exceeding the milder effect of `w/o Expand`, and it likewise yields a higher NPDI on four of the five datasets. This indicates that the hierarchy, by summarizing common configuration patterns shared across instances, is what endows HEG-Config with generalizable knowledge, while the neighbor expansion enriches the retrieved context with nearby experiences and contributes comparably less to the overall efficiency.

Strict prompting is essential for stable and efficient configuration. We ablate the distillation prompt, where the strict prompt denotes our designed template that forces experience into explicit feature-conditioned rules ($\phi(\mathbf{f}) \rightarrow \theta$) under a fixed output format, while the flexible prompt lifts these constraints and lets the model summarize freely (details in Appendix E.3). As Table 3 shows, the strict prompt outperforms the flexible one on every dataset: under the flexible prompt the NPDI exceeds 1 on several datasets and even fails to find a feasible solution on some instances, whereas the strict prompt keeps the NPDI below 1 on all five datasets. This is because the strict template compels the model to distill explicit feature-to-parameter rules that can be reliably retrieved and reused, whereas the free-form summaries from the flexible prompt lack such explicit associations, making them harder to match against a new instance and offering less concrete guidance once retrieved.

The aggregation threshold τ must balance information accumulation and hierarchy depth. Table 4 reports the effect of τ. τ governs the granularity of abstraction, and $\tau = 5$ – the setting adopted in our main experiments – achieves the best NPDI on all five datasets. The sensitivity to τ reflects a trade-off: a small threshold aggregates too frequently, so each high-level summary is built from very few lower-level nodes and inherits limited information; a large threshold aggregates rarely, producing a hierarchy that is too shallow to capture distinct levels of abstraction. Notably, $\tau = 10$ degrades performance more severely than $\tau = 2$, indicating that preserving a sufficiently deep hierarchy matters more than avoiding overly frequent aggregation.

4.4 More Analyses

Impact of hierarchy levels. Based on our observations from the ablation study, we further analyze the contribution of different hierarchy levels, and the results are reported in Table 5, with the

[2]In the relative gap, every instance for which no feasible solution is found contributes 10,000 (Section 4.1).

following findings. (1) **The full multi-level hierarchy achieves the best performance across all datasets.** Full (L0–L3) attains the lowest gap and NPDI on all five datasets, confirming that the complete multi-granularity experience structure is optimal. (2) **A single level is insufficient for reliable configuration.** Both L0 only and L3 only fail drastically: the gap on MIPLIB deteriorates sharply, and one instance fails to find a feasible solution on MIRP-medium. This indicates that relying on a single level, lacking experience at different granularities as reference, is insufficient. (3) **Incorporating higher-level experience progressively improves performance.** Along the sequence L0 only $\rightarrow$ L0–L1 $\rightarrow$ Full, the gap on MIPLIB drops sharply, and the NPDI likewise improves overall, suggesting that progressively enriching the context with higher-level experience is consistently beneficial. (4) **The base level is indispensable.** Discarding the base level while retaining only high-level summaries (L1–L3) raises the NPDI across all datasets and even fails to find a feasible solution on MIRP-medium, indicating that the fine-grained experience captured at the base level is irreplaceable.

More training instances generally improve configuration quality. The experience graph is constructed from the tuning trajectories of a set of training instances. Here we examine how the number of these instances affects HEG-Config's configuration quality. Figure 2 plots the NPDI as the training set grows from 20 to 125 instances. As the number of training instances grows, the NPDI generally decreases across all datasets. With very few instances, the NPDI exceeds 1 on several datasets, indicating configurations worse than the default. When the training set reaches 80 instances, the NPDI falls below 1 on every dataset and continues to decline on most, showing no sign of saturation at the largest setting tested. These results demonstrate that increasing the amount of training experience can further improve configuration quality.

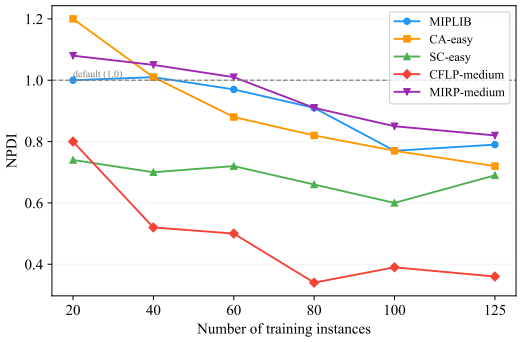


Figure 2: Influence of training instance count.

Additional Analyses. Complementary analyses are provided in the appendix. Appendix G.3 examines the impact of reducing the LLM size, showing that HEG-Config retains a clear advantage even with a smaller model. Appendix G.4 evaluates HEG-Config under different solving time budgets, and HEG-Config proves effective under all of them, with the mean NPDI decreasing as the budget grows. Appendix G.5 shows that the HEG-Config architecture also applies to Gurobi, keeping the NPDI below 1 on all datasets. Appendix G.6 analyzes HEG-Config's time overhead, revealing that it stems primarily from the retrieval backend; replacing it with a naive RAG backend substantially reduces the cost with little impact on performance.

5 Related Work

Due to the 9-page limit for the main paper at submission, we place the related work in Appendix A. If accepted, the limit increases to 10 pages, and we will move the related work into the main paper.

6 Conclusion

In this paper, we present HEG-Config, an LLM-guided framework that generates instance-specific parameter configurations for MIP solvers. HEG-Config distills historical tuning trajectories into experience cards, progressively extends them into a hierarchical experience graph, and retrieves multi-level experience to guide the LLM in generating an effective configuration for each new instance. Experiments on the heterogeneous MIPLIB benchmark and the cross-dataset D-MIPLIB evaluation show that HEG-Config consistently improves solving efficiency over the default configuration. Moreover, in the conventional homogeneous-mix setting, HEG-Config also attains the best performance. In future work, we plan to incorporate MIP instance features more deeply into the experience retrieval mechanism and investigate richer mechanisms for managing the experience.

AI USE STATEMENT

In this work, we used generative AI tools to assist with drafting and polishing parts of the manuscript and to aid in implementing and debugging portions of the experimental codebase. We have not used generative AI tools to fabricate or modify any experimental results; all reported results are produced and verified by our own experiments. The proposed method itself relies on a large language model as a core component, which is fully disclosed and described in the main text. We have reviewed all AI-assisted writing and code, and we take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work addresses the parameter configuration of mixed-integer programming solvers and does not involve human subjects, personal data, or sensitive applications. We foresee no direct negative societal or ethical impacts beyond the general considerations of improving the efficiency of optimization solvers.

REPRODUCIBILITY STATEMENT

The complete method, including experience distillation (Section 3.1), hierarchical experience graph construction (Section 3.2), and hierarchy-aware retrieval (Section 3.3), is described in the main text, with the detailed procedures given in the appendix (Algorithms 1, 2, and 3). The main experimental settings, including parameter selection, dataset and task partitioning, evaluation protocol, and baselines, are described in Section 4.1; more detailed information, such as the complete list of selected parameters (Appendix C.1), the dataset information (Appendix C.4), the prompt templates (Appendix E), and the baselines (Appendix C.3), is provided in the appendices. To further facilitate reproduction, we will release the source code upon acceptance.

REFERENCES

Tobias Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, Jul 2009. ISSN 1867-2957. doi: 10.1007/s12532-008-0001-1. URL https://doi.org/10.1007/s12532-008-0001-1.

Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *International Conference on Principles and Practice of Constraint Programming*, pp. 142–157. Springer, 2009.

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

Gérard Cornuéjols, Ranjani Sridharan, and Jean-Michel Thizy. A comparison of heuristics and relaxations for the capacitated plant location problem. *European journal of operational research*, 50(3):280–297, 1991.

Emily B Fisher, Richard P O'Neill, and Michael C Ferris. Optimal transmission switching. *IEEE transactions on power systems*, 23(3):1346–1355, 2008.

Luca Franceschi, Michele Donini, Valerio Perrone, Aaron Klein, Cédric Archambeau, Matthias Seeger, Massimiliano Pontil, and Paolo Frasconi. Hyperparameter optimization in machine learning. *Foundations and Trends® in Machine Learning*, 18(6):1054–1201, 2025. ISSN 1935-8245. doi: 10.1561/2200000088. URL http://dx.doi.org/10.1561/2200000088.

Maxime Gasse, Simon Bowly, Quentin Cappart, Jonas Charfreitag, Laurent Charlin, Didier Chételat, Antonia Chmiela, Justin Dumouchelle, Ambros Gleixner, Aleksandr M Kazachkov, et al. The machine learning for combinatorial optimization competition (ml4co): Results and insights. In *NeurIPS 2021 competitions and demonstrations track*, pp. 220–231. PMLR, 2022.

Alexander Georges, Ambros Gleixner, Gorana Gojic, Robert Lion Gottwald, David Haley, Gregor Hendel, and Bartlomiej Matejczyk. Feature-based algorithm selection for mixed integer programming. Technical Report 18-17, ZIB, Takustr. 7, 14195 Berlin, 2018.

Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 2021. doi: 10.1007/s12532-020-00194-3. URL https://doi.org/10.1007/s12532-020-00194-3.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. Lightrag: Simple and fast retrieval-augmented generation. *arXiv preprint arXiv:2410.05779*, 2024.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2026. URL https://www.gurobi.com.

Dorit S Hochbaum and Anu Pathria. Forest harvesting and minimum cuts: a new approach to handling spatial constraints. *Forest Science*, 43(4):544–554, 1997.

Christopher Hojny, Mathieu Besançon, Ksenia Bestuzheva, Sander Borst, João Dionísio, Johannes Ehls, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Adrian Göß, Alexander Hoen, Jacob von Holly-Ponientzietz, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Marco Lübbecke, Stephen J. Maher, Paul Matti Meinhold, Gioni Mexi, Til Mohr, Erik Mühmer, Krunal Kishor Patel, Marc E. Pfetsch, Sebastian Pokutta, Chantal Reinartz Groba, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Matthias Walter, Dieter Weninger, and Liding Xu. The scip optimization suite 10.0, 2025. URL https://arxiv.org/abs/2511.18580.

Abdelrahman Hosny and Sherief Reda. Automatic milp solver configuration by learning problem similarities, 2023. URL https://arxiv.org/abs/2307.00670.

Weimin Huang, Taoan Huang, Aaron M Ferber, and Bistra Dilkina. Distributional miplib: a multi-domain library for advancing ml-guided milp methods, 2024.

F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stuetzle. Paramils: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, October 2009. ISSN 1076-9757. doi: 10.1613/jair.2861. URL http://dx.doi.org/10.1613/jair.2861.

Gabriele Iommazzo, Claudia D'Ambrosio, Antonio Frangioni, and Leo Liberti. *Algorithm Configuration Problem*, pp. 1–8. Springer International Publishing, 2022. ISBN 9783030546212. doi: 10.1007/978-3-030-54621-2_749-1. URL http://dx.doi.org/10.1007/978-3-030-54621-2_749-1.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. *CoRR*, abs/2005.11401, 2020. URL https://arxiv.org/abs/2005.11401.

Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pp. 66–76, 2000.

Hongpei Li, Ziyan He, Yufei Wang, Wenting Tu, Shanwen Pu, Qi Deng, and Dongdong Ge. Benloc: A benchmark for learning to configure mip optimizers, 2025. URL https://arxiv.org/abs/2506.02752.

Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pp. 661–670, 2010.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization, 2022. URL https://arxiv.org/abs/2109.09831.

Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. *arXiv preprint arXiv:2402.03921*, 2024.

Kanan Mahammadli and Seyda Ertekin. Sequential large language model-based hyper-parameter optimization. *arXiv preprint arXiv:2410.20302*, 2024.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025.

Dimitri J Papageorgiou, George L Nemhauser, Joel Sokol, Myun-Seok Cheon, and Ahmet B Keha. Mirplib–a library of maritime inventory routing problem instances: Survey, core model, and benchmark results. *European Journal of Operational Research*, 235(2):350–366, 2014.

Madeleine Pollack, Ryan Piansky, Swati Gupta, Alyssa Kody, and Daniel Molzahn. Equitably allocating wildfire resilience investments for power grids: The curse of aggregation and vulnerability indices. *arXiv preprint arXiv:2404.11520*, 2024.

Elias Schede, Jasmin Brandt, Alexander Tornede, Marcel Wever, Viktor Bengs, Eyke Hüllermeier, and Kevin Tierney. A survey of methods for automated algorithm configuration. *Journal of Artificial Intelligence Research*, 75:425–487, October 2022. ISSN 1076-9757. doi: 10.1613/jair.1.13676. URL http://dx.doi.org/10.1613/jair.1.13676.

Wen Song, Yi Liu, Zhiguang Cao, Yaoxin Wu, and Qiqiang Li. Instance-specific algorithm configuration via unsupervised deep graph clustering. *Engineering Applications of Artificial Intelligence*, 125:106740, 2023. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2023.106740. URL https://www.sciencedirect.com/science/article/pii/S0952197623009247.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM international conference on management of data*, pp. 1009–1024, 2017.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Lin Xu, Holger Hoos, and Kevin Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pp. 210–216, 2010.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Kristian Kersting, Jeff Z Pan, Hinrich Schütze, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*, 2025.

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*, 2023.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. *Advances in Neural Information Processing Systems*, 37:113519–113544, 2024a.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Minkai Xu, Joseph E Gonzalez, Bin Cui, and Shuicheng Yan. Supercorrect: Supervising and correcting language models with error-driven insights. *arXiv preprint arXiv:2410.09008*, 9, 2024b.

Ling Yang, Zhaochen Yu, Bin Cui, and Mengdi Wang. Reasonflux: Hierarchical llm reasoning via scaling thought templates. *arXiv preprint arXiv:2502.06772*, 2025.

Furong Ye, Diederick L. Vermetten, Carola Doerr, and Thomas Bäck. Non-elitist selection can improve the performance of irace, 2022. URL https://arxiv.org/abs/2203.09227.

Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, et al. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *Proceedings of the 2019 international conference on management of data*, pp. 415–432, 2019.

Michael R Zhang, Nishkrit Desai, Juhan Bae, Jonathan Lorraine, and Jimmy Ba. Using large language models for hyperparameter optimization. *arXiv preprint arXiv:2312.04528*, 2023.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731, 2024.

Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guchun Zhang, Kun Shao, Linyi Yang, et al. Memento: Fine-tuning llm agents without fine-tuning llms. *arXiv preprint arXiv:2508.16153*, 2025.

A Related Work

Algorithm Configuration. Traditional algorithm configuration (AC) aims to find a single best configuration over a set of training instances, typically assuming that these instances are drawn from one homogeneous distribution. Existing AC methods include Bayesian optimization (Van Aken et al., 2017; Lindauer et al., 2022), heuristics (Hutter et al., 2009; Ansótegui et al., 2009; Ye et al., 2022), and reinforcement learning (Zhang et al., 2019; Li et al., 2010). However, real-world MIP instances are highly heterogeneous, and a single configuration is unlikely to suit all of them, which has motivated instance-specific AC: instead of seeking one configuration for all, it selects an appropriate configuration for each instance individually. A representative line is algorithm portfolio methods (Xu et al., 2010; Georges et al., 2018), which maintain a set of candidate configurations and, for each incoming instance, choose the one predicted to perform best. Other learning-based methods (Hosny & Reda, 2023; Song et al., 2023) first explore the configuration space to select promising candidates, then train a model to map a new instance to one of these candidates. Such instance-specific methods can adapt the configuration to the specific properties of each instance, which is valuable when instances differ considerably. However, they are confined to an exploration-derived candidate set and rely on trained models; when target instances fall outside the training distribution or the whole dataset is highly heterogeneous, they may generalize poorly, and are therefore typically validated on a limited set of relatively homogeneous instance classes.

Large Language Models for Algorithm Configuration. Leveraging the reasoning capabilities of large language models (LLMs), recent work has begun to integrate LLMs into configuration pipelines. LLAMBO (Liu et al., 2024) and SLLMBO (Mahammadli & Ertekin, 2024) use LLMs to refine configurations within a search procedure, and other studies explore LLMs for understanding hyperparameter semantics (Zhang et al., 2023) and as iterative optimizers (Yang et al., 2023). These methods enhance the Traditional AC configuration search process with LLM reasoning, but the LLM essentially serves as an accelerating component within the existing search framework.

Memory-Augmented Reasoning and Experience Buffers. Integrating external experience buffers or memory mechanisms has emerged as a key paradigm to address the ephemeral nature of LLM reasoning. Approaches like ReasoningBank (Ouyang et al., 2025) and A-Mem (Xu et al., 2025), among others (Wang et al., 2024; Zhong et al., 2024), construct repositories of reasoning trajectories or procedural workflows to enhance generalization and reuse. Further advancing this, frameworks such as Memory-R1 (Yan et al., 2025) and Memento (Zhou et al., 2025), among others (Yang et al., 2025; 2024b), employ RL or preference optimization to enable self-evolving thoughts and automated self-correction. Additionally, BoT (Yang et al., 2024a) introduces a mechanism to instantiate thought templates, further enhancing robustness in complex decision-making.

B MIP and SCIP Parameter Space

B.1 Structure of the SCIP Parameter Space

The parameter space of the SCIP solver exhibits high complexity and a hierarchical structure, comprising over 2,500 parameters in total.

Data Type Distribution: SCIP parameters are primarily categorized into six fundamental data types. Their statistical distribution and representative examples are summarized in Table 6.

Range Structure:

- **Discrete Categories**: Boolean parameters constitute the largest set of "switches" in the parameter space, strictly belonging to the binary domain $\{\text{TRUE}, \text{FALSE}\}$. Character-type parameters define a fixed set of categorical options. For instance, the range of `branching/scorefunc` is $\{\texttt{s}, \texttt{p}, \texttt{q}\}$ (representing sum, product, and quotient, respectively); despite their small cardinality, these parameters act as structural switches that alter core algorithmic logic. Integer and long-integer types are typically defined within a $[\min, \max]$ interval, often utilizing special values (e.g., -1) to denote physical meanings such as "unlimited" or "automatic".

Table 6: Distribution of SCIP Parameter Data Types

Type	Count	Nature	Typical Examples
`int`	∼1270	Discrete / Integer	Frequencies (`freq`), rounds, thread counts
`bool`	∼972	Discrete / Switch	Algorithm toggles (`enable`), execution checks
`real`	∼682	Continuous / Real	Decay factors, tolerances, weights (ranges typically $[0, 1]$)
`longint`	∼81	Discrete / Long	Node limits, search scale boundaries
`char`	∼58	Discrete / Char	Mode selection (e.g., `'s'` implex, `'d'` ual)
`string`	∼19	Text	External solver paths, file output formats

- **Continuous Space**: Real-type parameters are defined over continuous intervals. The majority are bounded within $[0, 1]$ (e.g., restart factors) or $[0, 1e + 20]$ (e.g., time limits), and are generally utilized to fine-tune heuristic preferences or convergence rates.

Visibility Hierarchy: SCIP stratifies its parameters into visibility levels. Standard parameters (`advanced: FALSE`) are those most frequently tuned by users, such as time limits, node limits, and basic strategy switches. Advanced parameters (`advanced: TRUE`) occupy the vast majority of the configuration space. These parameters delve into specific implementation details (e.g., LP basis matrix update frequencies, specific cut generation mechanics) and are typically reserved for expert-level tuning.

Logical Namespace Structure: Parameters are organized hierarchically via forward slashes ('/'), reflecting the solver's modular design. Major namespaces include: `branching/` (branch-and-bound logic), `constraints/` (constraint handler behavior), `heuristics/` (primal heuristic calls), `lp/` (underlying linear programming configuration), `presolving/` (preprocessing logic), and `separating/` (cut plane generation).

B.2 Performance Evaluation Metric for MIP

Evaluating MIP solver performance requires balancing solving speed and solution quality. Common metrics include runtime, optimality gap, and the primal-dual integral (PDI), with different metrics being more appropriate depending on instance difficulty.

For easy instances that can be solved to optimality within the time limit, runtime is the most direct and effective metric—shorter is better.

For hard instances where proving optimality within the time limit is infeasible, comparing only final runtime or success/failure discards valuable information. In such cases, the relative optimality gap and the PDI provide a more comprehensive characterization of the solving process.

The relative optimality gap follows the official SCIP definition (Achterberg, 2009):

$$\text{Gap} = \begin{cases} 0, & \text{if } P = D, \\ \dfrac{|P - D|}{\min(|P|, |D|)}, & \text{if } P \cdot D > 0, \\ \infty, & \text{if } P \cdot D \leq 0, \end{cases} \tag{6}$$

where P denotes the primal upper bound (best feasible solution) and D the dual lower bound. When P and D share the same sign, the gap is normalized to mitigate sensitivity to objective scaling; when they have opposite signs or one is zero, the gap is defined as infinity, indicating that the bounds straddle zero and meaningful normalization is not possible.

We note that when the primal bound P and dual bound D have opposite signs (or one is zero), SCIP defines the gap as infinity. However, after thorough inspection of solver logs across all experimental

runs, we found that P **and** D **always maintained the same sign** (i.e., $P \cdot D > 0$) in every case. Consequently, the gap remained finite throughout all experiments and can be safely used for numerical comparison. Therefore, we adopt SCIP's relative gap definition in our experiments and use it as one of the core metrics for final performance evaluation. For a dataset, we report the sum of the gaps over all its instances. Additionally, to facilitate presentation, we assign a gap of $10{,}000$ to instances for which no feasible solution is found within the time limit, so that the reported gap directly reflects the number of such failures.

However, the gap captures only the final state and ignores the convergence dynamics. To address this, we adopt the primal-dual integral (PDI), defined as the integral of the relative gap over the overall running time:

$$\text{PDI} = \int_0^T \text{gap}(t)\, dt, \tag{7}$$

where $\text{gap}(t)$ denotes the relative gap at time t.

PDI is a holistic metric: it simultaneously rewards rapid discovery of high-quality feasible solutions (lowering $P(t)$) and swift improvement of the dual bound (raising $D(t)$). Minimizing the PDI implies maintaining tighter primal-dual bounds throughout the search, leading to more robust and efficient solving behavior.

To compare solving efficiency across methods, we define the normalized primal-dual integral (NPDI) as our main evaluation metric. Given a dataset, the NPDI of a method is the ratio between the summed primal-dual integral over all instances under that method and the summed primal-dual integral under the default configuration:

$$\text{NPDI} = \frac{\sum_i \text{PDI}_i}{\sum_i \text{PDI}_i^{\text{default}}}, \tag{8}$$

where PDI_i is the primal-dual integral of instance i under a given method and $\text{PDI}_i^{\text{default}}$ that under the default configuration. An NPDI below 1 indicates higher solving efficiency than the default.

During the offline construction phase, we use the **PDI as the optimization objective for Bayesian Optimization (BO)**, as it provides a continuous and informative feedback signal suitable for surrogate modeling; in the testing phase, performance is measured by the NPDI.

B.3 MIPLIB FEATURE EXTRACTOR

the official **feature_extractor** from **MIPLIB 2017** generates a fixed-length 147-dimensional feature vector categorized into five distinct groups:

- **Variable Statistics**: Captures problem scale, the distribution of binary/integer/continuous variables, and boundary properties;
- **Objective Function Features**: Characterizes the dynamic range and statistical distribution of objective coefficients;
- **Constraint Structure**: Identifies the proportion of special constraint substructures (e.g., knapsack, set cover) and the density of the constraint matrix;
- **Matrix Numerical Features**: Includes the mean, variance, and sparsity patterns of constraint coefficients;
- **Graph Features**: Extracts topological information, such as cliques and degree distributions, using the variable-constraint incidence graph.

C MORE DETAILS ABOUT EXPERIMENT SETTING

C.1 SELECTION OF KEY PARAMETER SUBSET

We select 25 key parameters that directly shape the search tree morphology, cut generation intensity, and feasible solution discovery, making them the primary drivers of solver performance.

- **Presolving**: We regulate the aggressiveness of problem reduction and restart strategies, effectively determining the model compression rate prior to the main search.
- **Root Node Processing & Separating**: Parameters in this category control the tightening intensity of the initial LP relaxation and cut plane generation. They dictate the initial search quality and constraint strength, directly influencing the quality of the global lower bound.
- **Branching Strategies**: Core settings governing scoring functions, priority of branching rules (e.g., fullstrong), and binary variable preferences determine the morphology and growth direction of the search tree.
- **Node Selection**: Defines the strategy for choosing the next node to process, orchestrating the exploration of the search tree.
- **Primal Heuristics**: By adjusting the calling frequencies of various heuristic algorithms (including ALNS, crossover, feaspump, etc.) and tuning their active states and node budgets, we balance the computational allocation between finding high-quality feasible solutions (upper bounds) and proving optimality.
- **LP Solving**: This aspect defines the solver's core execution efficiency. Optimizing pricing strategies balances the cost per Simplex step against convergence speed, while internal scaling and presolving settings offer flexibility between numerical stability and computational overhead.

The detailed list of parameters and their functional descriptions are presented in Table 7.

C.2 Gurobi Parameter Selection

To evaluate the applicability of HEG-Config to solvers beyond SCIP, we additionally conduct experiments on Gurobi, a leading commercial MIP solver. We select 15 parameters that we consider important for MIP solving performance, spanning the core components of the solver, namely presolving, cutting planes, heuristics, and branching. Table 8 lists these parameters together with their types, ranges, defaults, and descriptions. For Gurobi, the data collection and experience distillation pipeline is identical to that of SCIP.

C.3 Baselines

We compare HEG-Config against three categories of configuration methods:

- **Hydra** (Georges et al., 2018): a feature-based algorithm selection method that trains pairwise comparators over a portfolio of candidate configurations and aggregates their predictions through voting to produce a full ranking of the configurations. We adopt the Hydra implementation provided by (Georges et al., 2018). The candidate portfolio consists of 14 configurations, obtained by varying four groups of SCIP parameters—emphasis, heuristics, presolving, and separating—across settings such as aggressive, fast, off, and specific emphasis modes (see Table 9). This implementation also provides six variants based on different base learners: random forest (rf), k-nearest neighbors (knn), decision tree (dt), support vector classifier (svc), linear support vector classifier (lsvc), and logistic regression (lr).
- **MILPTune** (Hosny & Reda, 2023): a deep-metric-learning-driven instance-aware MILP solver configuration approach. It trains a graph-convolutional instance encoder to learn an embedding space where embedding proximity correlates with the primal-bound solution cost of problem instances. At inference time, for an unseen MILP instance, the model maps it into the learned latent space, retrieves nearest-neighbor solved instances, and reuses their well-explored solver hyperparameter configurations to obtain high-quality solutions without expensive online configuration search.
- **KNN**: a nearest-neighbor method that represents instances using the official MIPLIB 2017 feature extractor (Gleixner et al., 2021), and retrieves the nearest training instance ($k = 1$) in the feature space.

For the two nearest-neighbor methods (MILPTune and KNN), the recommended configuration for a new instance is obtained by retrieving the nearest training instance and directly outputting that

Table 7: Selected 25 Key SCIP Parameters and Their Functional Roles

Category & Parameter	Description & Impact
Presolving (2 parameters)	
`presolving/maxrounds`	Controls maximum rounds for preprocessing.
`presolving/restartfac`	Adjusts the restart factor, balancing initial resource investment against problem reduction.
Separating / Cut Generation (4 parameters)	
`separating/maxrounds`	Restricts cut plane generation rounds per node.
`separating/maxroundsroot`	Restricts cut plane generation rounds at the root node.
`separating/maxcuts`	Restricts the number of cuts applied per round.
`separating/maxcutsroot`	Restricts the number of cuts applied per round at the root node.
Branching Strategies (4 parameters)	
`branching/scorefunc`	Determines how variable importance is evaluated (scoring method).
`branching/preferbinary`	Decides whether to prioritize 0-1 variables. Directly shapes the search tree.
`branching/fullstrong/priority`	Sets priority for the full strong branching rule.
`branching/relpscost/sbiterquot`	Controls maximal fraction of strong branching LP iterations compared to node relaxation.
Node Selection (2 parameters)	
`nodeselection/estimate/stdpriority`	Priority of node selection rule based on the objective best estimate.
`nodeselection/restartdfs/stdpriority`	Priority of node selection rule for restart depth-first search.
Primal Heuristics (10 parameters)	
`heuristics/alns/freq`	Controls the calling frequency for ALNS.
`heuristics/alns/nodesquot`	Configures fraction of nodes budget for ALNS.
`heuristics/alns/crossover/active`	Configures whether the ALNS crossover neighborhood is active.
`heuristics/alns/mutation/active`	Configures whether the ALNS mutation neighborhood is active.
`heuristics/crossover/freq`	Controls the calling frequency of crossover heuristic.
`heuristics/feaspump/freq`	Controls the calling frequency of feaspump heuristic.
`heuristics/localbranching/freq`	Controls the calling frequency of localbranching heuristic.
`heuristics/mutation/freq`	Controls the calling frequency of mutation heuristic.
`heuristics/rens/freq`	Controls the calling frequency of rens heuristic.
`heuristics/rins/freq`	Controls the calling frequency of rins heuristic.
LP Solving (3 parameters)	
`lp/pricing`	Controls pricing strategy, affecting LP iteration cost.
`lp/presolving`	Internal LP preprocessing settings.
`lp/scaling`	Internal LP scaling configurations.

instance's best configuration. This best configuration is obtained during data collection, where TPE samples 50 configurations for each training instance and the one with the lowest primal-dual integral is retained as the optimal configuration of that instance.

C.4 Dataset Details

Our benchmark datasets include MIPLIB 2017 and the **Distributional MIPLIB (D-MIPLIB)** benchmark (Huang et al., 2024). MIPLIB 2017 can be obtained from `https://miplib.zib.de/downloads/benchmark.zip`, and D-MIPLIB is available at `https://huggingface.co/datasets/weiminhu/D-MIPLIB`. Next, we introduce D-MIPLIB in detail, specifically focusing on the following core domains provided by the dataset:

Table 8: Selected Gurobi parameters and their descriptions.

Parameter	Type	Range	Default	Description
`Presolve`	int	[-1, 2]	-1	Controls the presolve level. -1: automatic, 0: off, 1: conservative, 2: aggressive. Aggressive presolve can reduce the model size more but costs more time.
`Aggregate`	int	[0, 2]	1	Presolve aggregation level. 0: off, 1: default, 2: aggressive. Aggregation combines/substitutes variables during presolve.
`PreSparsify`	int	[-1, 2]	-1	Presolve sparsify reduction. -1: automatic, 0: off, 1/2: increasingly on. Can reduce the number of nonzeros in the constraint matrix, useful for dense models.
`PreCrush`	int	[0, 1]	0	Allows presolve to translate constraints on the original model to equivalent constraints on the presolved model. 0: off, 1: on. Needed when using user cuts.
`Cuts`	int	[-1, 3]	-1	Global cutting-plane aggressiveness. -1: automatic, 0: off, 1: moderate, 2: aggressive, 3: very aggressive. More cuts tighten the relaxation but slow each node.
`CutPasses`	int	[-1, 2000000000]	-1	Maximum number of cutting-plane passes performed at the root node. -1: automatic (chosen by Gurobi). Larger values allow more rounds of cut generation.
`Heuristics`	real	[0, 1]	0.05	Fraction of runtime spent on feasibility heuristics. 0: off. Higher values find incumbents earlier (improving the primal bound / PD integral) but leave less time for search.
`RINS`	int	[-1, 2000000000]	-1	Frequency of the RINS (Relaxation Induced Neighborhood Search) heuristic. -1: automatic, 0: off, n: run every n nodes. Effective for improving incumbents on hard MIPs.
`PumpPasses`	int	[-1, 2000000000]	-1	Number of feasibility-pump passes. -1: automatic, 0: off. Useful for finding an initial feasible solution when the model is hard to satisfy.
`ImproveStartNodes`	real	[0, 1e100]	1e100 (infinity)	Node count at which Gurobi switches to a solution-improvement strategy (focus on finding better incumbents rather than proving optimality). Default infinity means never switch by node count. Lower values shift focus to the primal bound earlier.
`SubMIPNodes`	int	[0, 2000000000]	500	Node budget for sub-MIP based heuristics (e.g. RINS). Larger budgets let those heuristics search harder for better incumbents at extra cost.
`MIPFocus`	int	[0, 3]	0	High-level solve focus. 0: balanced, 1: find feasible solutions quickly (better primal bound early), 2: prove optimality (better dual bound), 3: improve the best objective bound. Choose based on whether the bottleneck is primal or dual.
`VarBranch`	int	[-1, 3]	-1	Branch-variable selection rule. -1: automatic, 0: pseudo reduced-cost, 1: pseudo shadow-price, 2: maximum infeasibility, 3: strong branching. Strong branching reduces node count but is expensive per node.
`NodeMethod`	int	[-1, 5]	-1	Algorithm used to solve node relaxations (MIP). -1: automatic, 0: primal simplex, 1: dual simplex, 2: barrier. Dual simplex is usually best for warm-started node LPs.
`Symmetry`	int	[-1, 2]	-1	Symmetry detection level. -1: automatic, 0: off, 1: conservative, 2: aggressive. Exploiting symmetry helps on highly symmetric models (e.g. assignment/scheduling).

C.4.1 COMBINATORIAL AUCTIONS (CA)

The Combinatorial Auction (CA) problem models the winner determination process, where n bids (B_i, p_i) are given for subsets of m items, with $B_i \subseteq [m]$ denoting the item bundle and p_i the associated price, based on the CATS suite (Leyton-Brown et al., 2000). The objective is to maximize

Table 9: The 14 configurations evaluated by Hydra.

Parameter	Setting
emphasis	easycip
emphasis	feasibility
emphasis	hardlp
emphasis	optimality
heuristics	aggressive
heuristics	fast
heuristics	off
presolving	aggressive
presolving	fast
presolving	off
separating	aggressive
separating	fast
separating	off
default	default

the total revenue while ensuring that each item is allocated to at most one accepted bid.

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} p_i x_i \\ \text{s.t.} \quad & \sum_{i:j\in B_i} x_i \le 1, \quad \forall j \in [m], \\ & x_i \in \{0,1\}, \quad \forall i \in [n]. \end{aligned}$$

where x_i is a binary decision variable indicating whether bid i is accepted.

C.4.2 Set Covering (SC)

In a Set Covering (SC) instance, we are given m elements indexed by $i \in [m]$ and a collection S of n sets whose union covers all elements. The goal is to select a minimum number of sets from S such that the union of the selected sets still covers all elements. It is widely applied in scheduling and crew pairing (Balas & Ho, 2009).

$$\begin{aligned} \min \quad & \sum_{s\in S} x_s \\ \text{s.t.} \quad & \sum_{s\in S:i\in s} x_s \ge 1, \quad \forall i \in [m], \\ & x_s \in \{0,1\}, \quad \forall s \in S. \end{aligned}$$

where x_s is a binary decision variable indicating whether set s is selected.

C.4.3 Generalized Independent Set Problem (GISP)

The Generalized Independent Set Problem (GISP) extends the independent set problem by allowing the removal of selected edges at a given cost to relax adjacency constraints (Hochbaum & Pathria, 1997). Given an undirected graph $G = (V, E)$ and a subset of removable edges $E' \subseteq E$, each vertex $v \in V$ has a revenue r^V, and each removable edge $e \in E'$ incurs a deletion cost c^E. The objective is to select a set of vertices and remove some edges so as to maximize the total vertex revenue minus edge deletion costs, while ensuring that the selected vertices form an independent set in the modified

graph.

$$\begin{aligned}
\max \quad & \sum_{v \in V} r^V x_v - \sum_{e \in E'} c^E y_e \\
\text{s.t.} \quad & x_u + x_v \leq 1, \quad \forall (u,v) \in E \setminus E', \\
& x_u + x_v \leq 1 + y_{u,v}, \quad \forall (u,v) \in E', \\
& x_v \in \{0,1\}, \quad \forall v \in V, \\
& y_e \in \{0,1\}, \quad \forall e \in E'.
\end{aligned}$$

where x_v indicates whether vertex v is selected, and y_e indicates whether removable edge e is deleted.

C.4.4 Capacitated Facility Location Problem (CFLP)

CFLP aims to select a subset of facilities to open and determine shipment quantities from opened facilities to customers, satisfying all demands while minimizing total cost (fixed opening cost + transportation cost).

$$\begin{aligned}
\min \quad & \sum_{f \in F} c^s_f x_f + \sum_{c \in C} \sum_{f \in F} c^t_{c,f} y_{c,f} & \\
\text{s.t.} \quad & \sum_{f \in F} y_{c,f} = d_c, & \forall c \in C \\
& \sum_{c \in C} y_{c,f} \leq b_f x_f, & \forall f \in F \\
& y_{c,f} \geq 0, & \forall c \in C, f \in F \\
& x_f \in \{0,1\}, & \forall f \in F
\end{aligned}$$

where: C: The set of customers. F: The set of facilities. d_c: Demand required by customer c. c^s_f: Startup cost defined for facility f. $c^t_{c,f}$: Transportation cost per unit shipped from facility f to customer c. b_f: Capacity budget (maximum capacity) defined for facility f. $x_f \in \{0,1\}$: Binary variable indicating whether facility f is opened. $y_{c,f} \geq 0$: Shipment quantity from facility f to customer c. Test instances are often randomly generated according to (Cornuéjols et al., 1991).

C.4.5 Maritime Inventory Routing Problem (MIRP)

The Maritime Inventory Routing Problem (MIRP) arises from global bulk shipping and involves both ship routing and inventory management. The problem determines routes from production ports to consumption ports to minimize transportation costs while managing the inventory at these ports. (Papageorgiou et al., 2014) introduced a mixed-integer programming model of this problem class and developed a library of publicly available test instances (MIRPLib). The MIRP instances in Distributional MIPLIB were curated from the test set used in the Machine Learning for Combinatorial Optimization (ML4CO) (Gasse et al., 2022) competition. The data follows the established train, validation, and test split as defined in the ML4CO benchmark.

C.4.6 Optimal Transmission Switching (OTS)

In energy planning, the Optimal Transmission Switching (OTS) (Fisher et al., 2008) problem under high wildfire ignition risk is a subset of network topology optimization problems. Transmission grids are represented as a series of power buses (vertices) connected by power lines (edges). During high wildfire ignition risk, methods to mitigate the risk of transmission lines starting wildfires include de-energizing or undergrounding the lines. De-energizing lines removes the ignition risk but interrupts power delivery, whereas undergrounding lines allows safe power delivery at a higher cost. This problem examines the optimal way to de-energize or underground transmission lines to reduce wildfire risk while minimizing power outages within a resource budget. The MILP formulation for the OTS instances used in this study is described in (Pollack et al., 2024).

D ALGORITHM OF HIERARCHICAL GRAPH

Algorithm 1 Part I: Hierarchical Experience Graph Construction

Input: Data Stream $\mathcal{D}$, Topology Function $\Phi(\cdot)$
Initialize: Hierarchy $\mathcal{V}_k \leftarrow \emptyset, \mathcal{E}_k \leftarrow \emptyset \; \forall k, \mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ where $\mathcal{E} = \bigcup \mathcal{E}_i, \mathcal{V} = \bigcup \mathcal{V}_i$
for each experience card $d_i \in \mathcal{D}$ **do**
  Create node v_i with content d_i; $\mathcal{V}_0 \leftarrow \mathcal{V}_0 \cup \{v_i\}$, level$(v_i) = 0$
  ▶ **Local Topology Linking**
  $\mathcal{N}_{new} \leftarrow \{u \in \mathcal{V}_0 \setminus \{v_i\} \mid u \text{ is highly relevant to } v_i \text{ via } \Phi\}$
  for each $u \in \mathcal{N}_{new}$ **do**
    $w \leftarrow \Phi(v_i, u), \mathcal{E}_0 \leftarrow \mathcal{E}_0 \cup \{(v_i, u, w)\}$
  end for
  ▶ **Recursive Abstraction Trigger**
  Call Algorithm 2$(v_i, 0)$
end for
Return $\mathcal{G}$

Algorithm 2 Recursive Restructuring Graph

Input: Target Node v, Level k
Hyperparameter: Aggregation threshold τ
Context: $\mathcal{G}$, Adjacency Function $\mathcal{N}(\cdot)$
$\mathcal{L}(v) \leftarrow \{u \in \mathcal{N}(v) \mid \text{level}(u) = k \wedge \neg\text{summarized}(u)\}$
$\mathcal{S}_v \leftarrow \mathcal{L}(v) \cup \{v\}$
▶ **Density Check**
if $|\mathcal{S}_v| \geq \tau$ **then**
  $c \leftarrow v$ {v serves as the neighborhood pivot}
else
  $\mathcal{C}_{cand} \leftarrow \{u \in \mathcal{L}(v) \mid |\mathcal{S}_u| \geq \tau\}$
  if $\mathcal{C}_{cand} \neq \emptyset$ **then**
    $c \leftarrow \arg\max_{u \in \mathcal{C}_{cand}} w(v, u)$
  else
    return {Density does not meet τ, terminate recursion}
  end if
end if
▶ **Summarization & Folding**
$\mathcal{S}_c \leftarrow \mathcal{L}(c) \cup \{c\}$
$p \leftarrow$ LLM_Summarize($\{\text{content}(u) \mid u \in \mathcal{S}_c\}$), child_list$(p) \leftarrow \mathcal{S}_c$, level$(p) = k + 1$
for each $u \in \mathcal{S}_c$ **do**
  summarized$(u) \leftarrow$ True; parent$(u) \leftarrow p$
end for
$\mathcal{V}_{k+1} \leftarrow \mathcal{V}_{k+1} \cup \{p\}$
▶ **Inter-node Edge Projection**
for each $q \in \mathcal{V}_{k+1} \setminus \{p\}$ **do**
  $\Omega \leftarrow \{(u, z) \mid u \in \mathcal{S}_c, z \in \text{child_list}(q), (u, z) \in \mathcal{E}_k\}$
  if $\Omega \neq \emptyset$ **then**
    $w_{pq} \leftarrow \frac{1}{|\Omega|} \sum_{(u,z) \in \Omega} w(u, z), \mathcal{E}_{k+1} \leftarrow \mathcal{E}_{k+1} \cup \{(p, q, w_{pq})\}$
  end if
end for
▶ **Upward Propagation**
Call Algorithm 2$(p, k + 1)$

Algorithm 3 Part II: Hierarchy-Aware Experience Retrieval

Input: Query q, Hierarchy $\mathcal{G}$, Expansion Policy $\Psi(k)$
Output: Structured Context $\mathcal{R}_{final}$
► **Initial Seed Retrieval**
$\mathcal{R}_0 \leftarrow \text{BASERETRIEVE}(q, \mathcal{V}_0)$
$\mathcal{R}_{all} \leftarrow \mathcal{R}_0$, Visited $\leftarrow \mathcal{R}_0$
► **Bottom-Up Path Propagation**
for each node $v \in \mathcal{R}_0$ **do**
  $curr \leftarrow v$
  while parent$(curr)$ exists **and** parent$(curr) \notin$ Visited **do**
    $p \leftarrow$ parent$(curr)$, $k \leftarrow$ level(p)
    $\mathcal{R}_{all} \leftarrow \mathcal{R}_{all} \cup \{p\}$, Visited $\leftarrow$ Visited $\cup \{p\}$
    ► **Horizontal Expansion (Neighbor Discovery)**
    $T_k \leftarrow \Psi(k)$ {Level-specific expansion budget}
    $\mathcal{N}_{\text{top}} \leftarrow \text{Top}_{T_k}(\{u \in \mathcal{V}_k \mid (p, u) \in \mathcal{E}_k \setminus \text{Visited}\})$ by edge weight
    $\mathcal{R}_{all} \leftarrow \mathcal{R}_{all} \cup \mathcal{N}_{\text{top}}$, Visited $\leftarrow$ Visited $\cup \mathcal{N}_{\text{top}}$
    $curr \leftarrow p$
  end while
end for
► **Knowledge Re-ranking & Fusion**
$\mathcal{R}_{sorted} \leftarrow \text{Sort}(\mathcal{R}_{all})$ {Priority: similarity to q and level specificity}
$\mathcal{R}_{final} \leftarrow \text{FORMATPROMPT}(\text{GroupLevel}(\mathcal{R}_{sorted}))$
Return $\mathcal{R}_{final}$

E PROMPT SETTING

E.1 EXPERIENCE SUMMARIZATION PROMPTS

E.1.1 EXPERIENCE CARD DISTILLATION PROMPT

Design rationale. This prompt is designed to convert raw optimization runs (problem description, chosen SCIP parameters, and solver outcomes) into a reusable *experience card*. The key design choice is to force the model to express knowledge as *quantitative if–then rules* that map dataset features (e.g., variable/constraint counts, density, and ratios) to parameter actions. This reduces overfitting to a single instance and improves transfer across problems by anchoring advice to measurable properties rather than post-hoc results (e.g., gap). The prompt also explicitly separates *(i) rules*, *(ii) a concise summary*, and *(iii) error analysis/blacklists* so that the resulting memory supports both positive reuse and safe avoidance of failure modes.

Prompt.

Experience Card Distillation Prompt

```
You are an expert in optimization and SCIP parameter tuning.
Your task is to analyze the problem description, suggested
    parameters, and execution result, then distill a reusable
    experience card.

**CRITICAL GOAL**: The core of this experience card MUST be a set
    of **Quantitative Rules**.
- The rules MUST link **Dataset Features** (e.g., number of
    variables, constraints, density, ratios) to **Parameter Choices
    **.
- **FORBIDDEN**: Do NOT create rules that only link parameters to
    parameters (e.g., "If presolving/restartfac < 0.5, set it to
    0.6"). This is meaningless.
```

```
- **REQUIRED**: Rules must be in the form: "If <DATASET_FEATURE>
    satisfies <CONDITION>, then <PARAMETER> should be <VALUE/RANGE
    >".

**EMPHASIS ON RATIOS & GENERALIZATION**:
- **Consider Both Absolute and Relative Features**: Do not rely
    solely on one type. You MUST consider both **Absolute Numbers**
    (e.g., number of variables, constraints) and **Ratios/Relative
    Features** (e.g., constraints/variables ratio, density).
- **Why**: Absolute numbers define the scale (Small vs Large),
    while ratios define the structure (Sparse vs Dense). Both are
    critical for generalization.
- Example: You can combine them, e.g., "If variables > 10000 (Large
    ) AND density < 0.01 (Sparse), then..."

Produce the final answer in the following sections:

1. **Quantitative Rules** (The most important section, provide at
    least 5-7 rules):
    - Format: `If <DATASET_FEATURE_CONDITION>, then <
    PARAMETER_ACTION>: <REASON>`
    - **Condition**: Must be a specific property of the problem
    instance.
    - Good Examples: `binary variables > 10,000`, `constraints/
    variables ratio < 0.5`, `density < 0.001`, `presence of SOS1
    constraints`.
    - Bad Examples: `gap > 10%` (this is a result, not a feature),
    `presolving/maxrounds < 10` (this is a parameter).
    - **Action**: Must be a precise parameter setting or range.
    - Example: `set heuristics/freq to -1`, `set separating/
    maxrounds to [5, 10]`.
    - **Reason**: A brief technical explanation linking the feature
     to the parameter choice.

2. **Experience Summary**:
    - Problem Profile: 1-2 sentences describing the problem type
    and scale.
    - Key Strategy: One-sentence summary of the most effective
    approach.

3. **Error Analysis & Bad Trends** (CRITICAL):
    - Identify parameters that led to **Timeouts**, **Errors**, or
    **Empty Results**.
    - **Contextualize with Features**: You MUST link the failure to
     specific problem features.
        - Format: "When <DATASET_FEATURE_CONDITION> (e.g.,
    NVAR_ALL_MEAN > 40), avoid setting <PARAMETER> to <VALUE> (or >
    <VALUE>) because it causes <FAILURE_TYPE>."
        - Example: "When constraints > 20000, avoid setting
    separating/maxrounds > 20."
    - Summarize unreasonable parameter ranges that should be
    blacklisted under specific conditions.
```

E.1.2 Graph-Level High-Order Summarization Prompt

Design rationale. This prompt is designed to build a hierarchical memory over experience cards by repeatedly compressing a group of related Level-k cards into a more abstract Level-$(k+1)$ card. The design focuses on (1) *validation and filtering* (discarding overfitted or contradictory rules), (2) *synthesis and generalization* (merging similar rules to cover wider feature ranges and capturing order-of-magnitude effects), and (3) *format preservation* so that higher-level nodes remain compatible with downstream retrieval and rule application. By making this summarization recursive, the system can scale memory while improving signal-to-noise at higher levels.

Prompt.

Graph-Level High-Order Summarization Prompt

```
You are a High-Level Summarizer. Your goal is to synthesize a set
    of 'Experience Cards' into a more robust, generalized Higher-
    Level Experience (Level {k+1}).

### INPUT DATA
Below are several experience cards collected from similar problem
    solving attempts:
{joined_contents}

### TASK
Synthesize these experiences into one single, coherent High-Level
    Experience Card.
1. **Validation & Filtering**:
   - Cross-verify the 'Quantitative Rules'. Retain rules that
    appear across multiple cards or are logically consistent with
    global optimization principles.
   - **Discard** rules that seem too specific to a single problem
    instance (overfitted), logically incorrect, or contain
    contradictory advice without clear context.
2. **Synthesis & Generalization**:
   - Merge similar rules into a broader rule covering a wider range
     of dataset features.
   - For 'Order of Magnitude', identify if the strategy changes as
    the problem grows.
3. **Hierarchy Preservation**:
   - The resulting card should still follow the standard format:
     - **Quantitative Rules**: (Focus on multi-verified and
    generalized rules)
     - **Experience Summary**: (Synthesized problem profile and
    global strategy)
     - **Error Analysis & Bad Trends**: (Consolidated blacklist of
    risky parameter ranges)

Ensure the output is technical, concise, and prioritized by the
    reliability of the lessons learned.
```

E.2 Configuration Recommendation Prompt

Design rationale. This prompt is designed to recommend a strong parameter configuration for a given instance by leveraging the retrieved experience. The key design choices are: (1) providing the experience context explicitly (as "data tables") and placing it before the problem description to bias the model toward reuse, (2) emphasizing conservatism via default-parameter anchoring ("do not deviate too much"), and (3) enforcing a strict JSON-only output format to ensure the result is executable without extra parsing. The prompt also encourages abstention (empty JSON) when the model lacks strong justification, which is a safety mechanism against extreme, solver-breaking parameter suggestions.

Prompt.

Configuration Recommendation Prompt

```
There is some experience about solving MIP problem. You can refer
    to the experience to generate better parameters.

---Data tables---

{context}
```

```
======= Problem Features =======

{the_problem}

the following parameters are allowed to be changed:

{allowed_params}

**Your Task:**
Analyze the problem description above, the provided experience, and
     valid parameters to generate a JSON object containing parameter
     values that you believe will yield the best performance.

**NOTE**:
- Experience may be wrong for this instance - verify each borrowed
    parameter against the problem features and your SCIP domain
    knowledge.
- You have ONE chance: prefer a few justified changes over timid
    near-default tweaks. Keep a parameter at its default only when
    you have a specific reason. Deviate within sane ranges - avoid
    pathological extremes unless the features clearly call for it.
- If you want to keep a parameter at its default, you can simply
    not include it in the JSON object.
- Do NOT touch the reserved parameters (limits/time, lp/threads,
    limits/gap).

**REQUIRED OUTPUT FORMAT:**
Analyze first, then answer. In your analysis, connect this problem'
    s features to each parameter you choose, borrow, or reject, and
    state how that choice serves the objective. Then provide ONLY
    the chosen parameters as JSON: ```json {{ ... }} ```.
```

E.3 FLEXIBLE PROMPT

Design rationale. To analyze the role of prompt structure, we construct a more flexible prompt as an ablation variant. Unlike the strict prompts used in our main method, this prompt does not enforce strict output formats or reasoning structures, allowing the model to generate summaries in a more free-form manner. This design serves to evaluate the importance of structured prompting for stability and performance.

Prompt.

Flexible Prompt

```
You are a general problem-solving analyst. Your task is to review a
     summary of a problem-solving process-including the problem
    description, various attempted solutions, and their results-and
    to extract generalizable insights.

Your goal is to create a concise and reusable summary of the key
    lessons learned, which could be useful for tackling similar
    challenges in the future, regardless of the specific domain.

Please organize your findings into these general categories:

1.  **Key Discoveries and Core Principles**:
    *   What were the most important takeaways from this process?
    *   Describe the fundamental principles behind what made
    certain solutions succeed where others failed. Focus on the "
    what" and "why."
```

```
2.  **Profile of the Successful Strategy**:
    *   Briefly describe the approach that yielded the best results
    .
    *   What were the critical components of this strategy?

3.  **Summary of the Challenge**:
    *   In simple terms, what was the nature of the problem that
    was addressed? (e.g., "This was a large-scale resource
    allocation problem with complex constraints.")

4.  **Patterns to Avoid**:
    *   What approaches or configurations proved to be ineffective
    or counter-productive?
    *   Summarize any clear trends that led to negative outcomes.
```

F EXPERIENCE EXAMPLE

Our experience representation is structured into three logically coherent components, each serving a distinct role in capturing, organizing, and applying solver configuration knowledge for MIP instances.

The first component, **Quantitative Rules**, encodes deterministic mappings from measurable problem features–such as binary variable ratio, constraint matrix density, and objective sparsity–to concrete SCIP parameter configurations. These rules strictly follow a "feature-to-action" paradigm, avoiding parameter-to-parameter dependencies, and include technical justifications to ensure interpretability.

Quantitative Rules

```
# **Experience Card: Quantitative Parameter Tuning for Sparse Pure-
    Binary MIPs (e.g., 30n20b8)**

## 1. **Quantitative Rules** (*Based on problem structure $\to$
    parameter action, with technical reasoning*)

> **All rules link problem features to parameter choices. No
    parameter-to-parameter rules.**

| Rule | Condition | Action | Reason |

| **R1: Binary Dominance $\to$ Prefer Binary Branching** | `
    BIN_VARS / VARS > 0.95` (98.8% in this case) | `set branching/
    preferbinary to 1` | When >95% variables are binary, forcing
    binary branching reduces search tree explosion and improves node
     selection efficiency. Binary branches are more informative in
    such problems. |

| **R2: Sparse Matrix $\to$ Limit Separation Rounds** | `
    CONSTR_MATRIX_DENSITY < 0.05` (2.19%) AND `NVAR_ALL_STD > 1000`
    (3416.4) | `set separating/maxrounds to [1, 100]` AND `set
    separating/maxroundsroot to [1, 50]` | High constraint size
    variance + low overall density indicates few strong cuts exist.
    Excessive separation wastes CPU and can cause instability
    without improving bounds. |

| **R3: Objective Sparsity $\to$ Optimize Feasibility Heuristics**
    | `TOTAL_OBJNONZERO_VARS < 10` AND `OBJCOEFF_ALL_STD < 5` (2.67)
     | `set heuristics/feaspump/freq to [10, 100]` | With only 2
    nonzero objective coefficients, the problem is nearly
```

```
    unconstrained. Feasibility pump is critical to find a valid
    solution quickly; setting it too low or too high fails to
    trigger. |

...
```

The second component, **Experience Summary**, abstracts the core profile of the target problem class and articulates a high-level configuration strategy. For instance, for sparse pure-binary MIPs, it recommends prioritizing lightweight separation and aggressive presolving rather than blindly enabling all advanced solver features.

Experience Summary

```
## 2. **Experience Summary**

- **Problem Profile**: Pure-binary MIP (98.8% binary vars), sparse
    structure (~2.19% density), minimal objective cost variation (
    only 2 nonzero coefficients), high volatility in variable bounds
     (high std in upper bounds).
- **Key Strategy**: Prioritize **binary branching**, **lightweight
    separation**, **aggressive presolving**, and **moderate, stable
    heuristics**. Avoid over-engineered parameter settings -- **let
    structure drive performance**, not power.
```

The third component, **Error Analysis & Bad Trends**, documents parameter settings that empirically lead to solver failures under specific structural conditions. It provides safety constraints for the recommendation system by explicitly listing blacklisted parameter ranges, thereby preventing catastrophic configuration choices.

Error Analysis & Bad Trends

```
## 3. **Error Analysis & Bad Trends**

> **Specific failures linked to problem features. Blacklisted
    parameter settings under conditions.**

| Bad Trend | Condition | Parameter Value/Range | Failure Type |
    Why It Fails |

| **Empty primal solution (primal_bound = 1e+30)** | `branching/
    preferbinary = 0` AND `BIN_VARS / VARS > 0.95` | `branching/
    preferbinary=0` | Infeasibility or solution failure | When
    binary dominance is high, ignoring binary branching leads to
    poor node selection and early pruning of feasible paths. |

| **Dual bound collapse (gap = 1e+30)** | `separating/maxrounds >
    1000` AND `CONSTR_MATRIX_DENSITY < 0.05` | `separating/maxrounds
     > 1000` | **Divergent solver behavior** | Over-separation in
    sparse problems generates useless cuts, consumes memory, and
    stalls progress -- especially when cut quality is low. |

...

> **Blacklisted Parameter Ranges (Under Condition)**:

- `branching/relpscost/pscostweight < -1e10` $\to$ **when
    UPPERBOUNDS_STD > 100**
- `separating/maxrounds > 1000` $\to$ **when CONSTR_MATRIX_DENSITY
    < 0.05**
```

```
- `heuristics/feaspump/freq = 0 or > 1e6` $\to$ **when
    TOTAL_OBJNONZERO_VARS < 10**
- `lp/pricing = 'a', 'd', 'f'` $\to$ **when DYNAMISM_MEAN < 0.6**
- `presolving/maxrounds = 0 or -1` $\to$ **when UPPERBOUNDS_DENSITY
    > 0.9 and VARS > 1000**
```

This three-part experience card design supports interpretable LLM reasoning, efficient RAG retrieval, and serves as a structured knowledge unit for cross-instance generalization.

G More Experiment Results and Additional Analyses

Table 10: **Detailed results per dataset**. For each dataset we report the total primal-dual (PD) integral, the normalized PD integral (NPDI), the improvement over the default configuration, and the total optimality gap. The improvement is defined as $1 - \text{NPDI}$ and expressed as a percentage; since the NPDI is rounded for display, it is computed from the unrounded NPDI and can therefore differ slightly from 1 minus the NPDI listed in the table. Lower is better for the PD integral, NPDI, and gap; higher is better for the improvement. We mark **the best values** in bold and underline the second-best values. The default SCIP configuration is the reference, so its NPDI is 1 and its improvement is 0.

	MIPLIB 2017				Combinatorial Auctions (Easy)				Capacitated Facility Location Problem (Easy)			
	PD integral↓	NPDI↓	Improv.↑ (PDI, %)	gap↓	PD integral↓	NPDI↓	Improv.↑ (PDI, %)	gap↓	PD integral↓	NPDI↓	Improv.↑ (PDI, %)	gap↓
Default	1279736.42	1.00	0.00%	4404.62	23257.58	1.00	0.00%	0.00	44618.15	1.00	0.00%	0.00
Hydra-rf	1409592.09	1.10	-10.15%	34356.63	45572.59	1.96	-95.95%	0.00	43335.58	0.97	2.87%	**0.00**
Hydra-dt	1494453.80	1.17	-16.78%	57645.86	27155.86	1.17	-16.76%	0.00	55008.95	1.23	-23.29%	**0.00**
Hydra-lsvc	1423845.58	1.11	-11.26%	51930.35	10031.47	**0.43**	**56.87%**	0.00	42667.93	0.96	4.37%	**0.00**
Hydra-lr	1346476.45	1.05	-5.22%	40035.64	**9923.81**	**0.43**	**57.33%**	0.00	43283.53	0.97	2.99%	**0.00**
Hydra-knn	1379782.53	1.08	-7.82%	26685.97	10064.24	**0.43**	**56.73%**	0.00	53776.38	1.21	-20.53%	0.04
Hydra-svc	1367258.88	1.07	-6.84%	26685.36	10469.70	0.45	54.98%	0.00	54279.52	1.22	-21.65%	0.04
MILPTune	1165869.18	0.91	8.90%	40024.78	12973.99	0.56	44.22%	0.00	32866.31	0.74	26.34%	**0.00**
KNN	1191048.10	0.93	6.93%	4361.59	26614.99	1.14	-14.44%	0.00	36602.05	0.82	17.97%	0.01
HEG-Config	**1015405.67**	**0.79**	**20.66%**	**16.60**	16840.31	0.72	27.59%	0.00	**19380.57**	**0.43**	**56.56%**	**0.00**
	Generalized Independent Set Problem (Easy)				Optimal Transmission Switching (Easy)				Set Covering (Easy)			
	PD integral↓	NPDI↓	Improv.↑ (PDI, %)	gap↓	PD integral↓	NPDI↓	Improv.↑ (PDI, %)	gap↓	PD integral↓	NPDI↓	Improv.↑ (PDI, %)	gap↓
Default	222520.21	1.00	0.00%	0.00	305232.44	1.00	0.00%	0.00	16857.12	1.00	0.00%	0.00
Hydra-rf	**97854.38**	**0.44**	**56.02%**	0.00	453286.10	1.49	-48.51%	**0.00**	16489.57	0.98	2.18%	0.00
Hydra-dt	202448.15	0.91	9.02%	0.00	445555.00	1.46	-45.97%	0.45	5191.45	0.31	69.20%	0.00
Hydra-lsvc	181183.76	0.81	18.58%	0.00	1001483.21	3.28	-228.11%	0.43	**5095.67**	**0.30**	**69.77%**	0.00
Hydra-lr	509096.00	2.29	-128.79%	0.00	997917.92	3.27	-226.94%	0.44	5445.63	0.32	67.70%	0.00
Hydra-knn	473304.77	2.13	-112.70%	0.00	439838.53	1.44	-44.10%	0.49	16350.70	0.97	3.00%	0.00
Hydra-svc	370988.11	1.67	-66.72%	0.00	296911.26	0.97	2.73%	**0.00**	15459.12	0.92	8.29%	0.00
MILPTune	226541.41	1.02	-1.81%	0.00	442258.28	1.45	-44.89%	**0.00**	8350.70	0.50	50.46%	0.00
KNN	187145.47	0.84	15.90%	0.00	274319.53	0.90	10.13%	0.25	14263.23	0.85	15.39%	0.00
HEG-Config	167789.92	0.75	24.60%	0.00	**263819.05**	**0.86**	**13.57%**	**0.00**	11672.08	0.69	30.76%	0.00
	Combinatorial Auctions (Medium)				Capacitated Facility Location Problem (Medium)				Generalized Independent Set Problem (Medium)			
	PD integral↓	NPDI↓	Improv.↑ (PDI, %)	gap↓	PD integral↓	NPDI↓	Improv.↑ (PDI, %)	gap↓	PD integral↓	NPDI↓	Improv.↑ (PDI, %)	gap↓
Default	56141.20	1.00	0.00%	0.03	103803.61	1.00	0.00%	0.01	1961404.01	1.00	0.00%	7.07
Hydra-rf	124356.56	2.22	-121.51%	0.14	120907.78	1.16	-16.48%	0.01	**622533.78**	**0.32**	**68.26%**	**0.00**
Hydra-dt	56762.84	1.01	-1.11%	**0.01**	188733.15	1.82	-81.82%	0.21	2362401.10	1.20	-20.44%	12.44
Hydra-lsvc	42772.58	0.76	23.81%	**0.01**	160791.83	1.55	-54.90%	0.17	1849073.70	0.94	5.73%	5.11
Hydra-lr	43296.31	0.77	22.88%	**0.01**	161891.49	1.56	-55.96%	0.18	3882410.33	1.98	-97.94%	26.66
Hydra-knn	43132.95	0.77	23.17%	**0.01**	126304.29	1.22	-21.68%	0.21	4605425.73	2.35	-134.80%	61.11
Hydra-svc	46075.61	0.82	17.93%	**0.01**	116520.55	1.12	-12.25%	0.01	2379619.97	1.21	-21.32%	12.46
MILPTune	**37089.62**	**0.66**	**33.94%**	**0.01**	66011.56	0.64	36.41%	**0.00**	2167339.43	1.10	-10.50%	0.82
KNN	55607.84	0.99	0.95%	0.08	73151.38	0.70	29.53%	0.01	1589340.98	0.81	18.97%	2.78
HEG-Config	45590.22	0.81	18.79%	0.03	**37347.36**	**0.36**	**64.02%**	**0.00**	1862204.16	0.95	5.06%	6.69
	Maritime Inventory Routing Problem (Medium)				Optimal Transmission Switching (Medium)				Set Covering (Medium)			
	PD integral↓	NPDI↓	Improv.↑ (PDI, %)	gap↓	PD integral↓	NPDI↓	Improv.↑ (PDI, %)	gap↓	PD integral↓	NPDI↓	Improv.↑ (PDI, %)	gap↓
Default	982629.65	1.00	0.00%	10022.90	1619456.26	1.00	0.00%	4.64	147985.37	1.00	0.00%	0.00
Hydra-rf	959422.49	0.98	2.36%	45.33	3333545.75	2.06	-105.84%	11.36	146401.29	0.99	1.07%	**0.00**
Hydra-dt	977861.25	1.00	0.49%	20013.22	1626557.13	1.00	-0.44%	4.66	74573.52	0.50	49.61%	**0.00**
Hydra-lsvc	934816.69	0.95	4.87%	24.06	3765956.58	2.33	-132.54%	23.88	**59005.81**	**0.40**	**60.13%**	**0.00**
Hydra-lr	892624.39	0.91	9.16%	19.76	3745007.12	2.31	-131.25%	23.50	59167.27	**0.40**	**60.02%**	**0.00**
Hydra-knn	919182.25	0.94	6.46%	**18.15**	2890331.15	1.78	-78.48%	17.54	146507.06	0.99	1.00%	**0.00**
Hydra-svc	1000383.81	1.02	-1.81%	10025.74	1619199.67	1.00	0.02%	3.62	144364.05	0.98	2.45%	**0.00**
MILPTune	1260346.35	1.28	-28.26%	10022.77	3064011.27	1.89	-89.20%	26.16	133242.07	0.90	9.96%	0.06
KNN	1000015.63	1.02	-1.77%	34.37	2110315.44	1.30	-30.31%	11.38	183213.55	1.24	-23.81%	0.20
HEG-Config	**803257.98**	**0.82**	**18.25%**	19.53	**1336985.27**	**0.83**	**17.44%**	**3.41**	125885.22	0.85	14.93%	**0.00**

G.1 Detailed Results of the Main Experiment

Table 10 provides a detailed breakdown of Table 1 on each individual dataset, reporting the total primal-dual integral, the NPDI, the improvement over the default configuration, and the total relative optimality gap.

G.2 Evaluation under a Homogeneous-Mix Training Scenario

In our introduction, we note that existing instance-specific methods are typically developed and validated on mixtures of a few relatively homogeneous instance classes. To confirm that HEG-Config also performs well in this conventional setup, rather than only on the heterogeneous benchmark, we additionally construct an experience graph from a mixture of the CA-easy, SC-easy, CFLP-medium, and MIRP-medium datasets and evaluate each method on these same datasets. Table 11 reports the NPDI.

Table 11: **NPDI under a homogeneous-mix training scenario**. All methods are trained on a mixture of the CA-easy, SC-easy, CFLP-medium, and MIRP-medium datasets and evaluated on each of them. Lower values indicate better performance, with values below 1 indicating an improvement over the default. We mark **the best values** in bold and underline the second-best values.

	Hydra						MILPTune	KNN	HEG-Config
	rf	dt	lsvc	lr	knn	svc			
CA-easy	0.43	0.45	0.42	**0.41**	0.43	0.43	0.70	0.60	0.43
SC-easy	0.30	0.33	0.32	0.31	0.32	0.29	0.30	0.35	**0.28**
CFLP-medium	0.82	1.07	1.01	1.01	0.82	1.01	0.43	**0.30**	**0.30**
MIRP-medium	0.91	0.97	**0.79**	0.92	0.90	0.94	0.93	**0.79**	**0.79**
Mean	0.61	0.71	0.64	0.66	0.62	0.67	0.59	0.51	**0.45**

Even in this homogeneous-mix setup, HEG-Config attains the best average NPDI (0.45), staying below the second-best method (KNN, 0.51). While individual Hydra variants win on some datasets, no single method is consistently best, and HEG-Config ties or leads the strongest competitor on most datasets. This shows that HEG-Config's advantage is not confined to the heterogeneous setting, but persists even in the distribution that existing methods are typically designed for.

G.3 Performance with a Smaller LLM

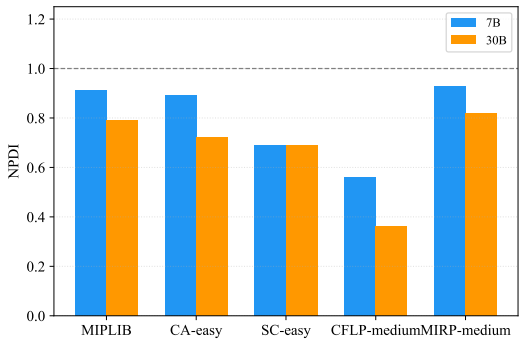


Figure 3: Performance comparison of HEG-Config on the 7B and 30B LLMs

We evaluate whether the configuration quality of HEG-Config depends strongly on the size of the underlying LLM, comparing a 7B and a 30B model in Figure 3.

HEG-Config remains effective with a substantially smaller model. The 30B model achieves a lower NPDI than the 7B model on four of the five datasets, while the 7B model still keeps the NPDI

well below 1 on all datasets. This indicates that HEG-Config retains a clear advantage over the default configuration even with a smaller model, supporting lightweight deployment in resource-constrained settings.

Table 12: NPDI of HEG-Config under different time limits.

	10	100	600	1000
MIPLIB	0.93	0.93	0.82	0.79
CA-easy	0.74	0.76	0.78	0.72
SC-easy	0.81	0.74	0.76	0.69
CFLP-medium	0.45	0.37	0.31	0.36
MIRP-medium	0.99	0.95	0.93	0.82
Mean	0.78	0.75	0.72	0.68

Table 13: NPDI of HEG-Config under the Gurobi and SCIP backends.

	gurobi	SCIP
MIPLIB	0.85	0.79
CA-easy	0.65	0.72
SC-easy	0.69	0.69
CFLP-medium	0.43	0.36
MIRP-medium	0.97	0.82
Mean	0.72	0.68

G.4 Robustness to Time Budget

We evaluate HEG-Config under different solving time budgets, and report the NPDI under time limits of 10/100/600/1000 seconds in Table 12.

HEG-Config configurations remain effective across different solving time budgets. Under all tested budgets, the NPDI stays below 1 on all datasets, and the mean NPDI decreases from 0.78 to 0.68 as the budget grows. This indicates that HEG-Config enables the solver to be more efficient under any time budget, with the advantage growing as the budget increases.

G.5 Applicability to Other Solvers

We evaluate whether the HEG-Config architecture generalizes beyond SCIP by applying it to Gurobi[3], a leading commercial MIP solver. Table 13 reports the NPDI under both backends.

The HEG-Config architecture is applicable across solver backends. The NPDI remains below 1 on all datasets under both SCIP and Gurobi, indicating that HEG-Config brings a substantial improvement over the default configuration on both. This confirms that the design of HEG-Config is not tied to a particular solver and can be applied to other backends.

Table 14: Time cost of different methods.

	rf	knn	dt	svc	lsvc	lr	MILPTune	KNN	HEG-Config	HEG-Config-NR
Data collection	6.46h	6.46h	6.46h	6.46h	6.46h	6.46h	24.11h	24.11h	24.11h	24.11h
Training / Experience construction	27.00s	13.00s	7.00s	7.00s	10.00s	8.00s	163s	0	6.28h	1435.1s
Inference per instance	1.91s	8.04s	0.47s	0.95s	0.30s	0.23s	1.9s	0.1s	15.6s	9.13s

[3]From Gurobi's parameters we select 15 important tunable parameters; the data collection and experience distillation pipeline is identical to that of SCIP. Details are provided in Appendix C.2.

Table 15: Construction time breakdown of HEG-Config.

Component	Time
Hierarchical experience construction	372s
LightRAG construction	6.18h

Table 16: NPDI of HEG-Config and HEG-Config-NR (naive RAG).

	HEG-Config	HEG-Config-NR
MIPLIB	0.79	0.83
CA-easy	0.72	0.75
SC-easy	0.69	0.68
CFLP-medium	0.36	0.37
MIRP-medium	0.82	0.85
Mean	0.68	0.70

G.6 Analysis of Time Overhead

We measure the time overhead of all baselines and HEG-Config, decomposed into data collection, training (or experience construction), and inference, as reported in Table 14. The training entry groups together the model training of traditional methods and the experience construction of HEG-Config for a unified comparison. The data collection time differs between Hydra and the remaining methods: Hydra uniformly takes 6.46h because it evaluates a fixed set of 14 configurations on every training instance and trains a 14-way selector, so the training data it requires is identical across all its variants. In contrast, KNN, MILPTune, and HEG-Config each take 24.11h because they all depend on the same process of sampling 50 configurations per instance through TPE; the distinction is that HEG-Config consumes the entire 50-configuration trajectory for experience distillation, whereas KNN and MILPTune only retain the best configuration among the 50 for nearest-neighbor matching.

We further investigate the experience construction time of HEG-Config, as reported in Table 15. This time splits naturally into two parts: constructing the base-level experience cards with a RAG backend (specifically LightRAG), and constructing the hierarchical experience graph from them. We directly time the hierarchical construction via timestamps and obtain the LightRAG portion by subtracting it from the total. The breakdown reveals that LightRAG consumes 6.18 hours of the 6.28-hour total, accounting for roughly 98% of the construction time, whereas the hierarchical construction itself takes only 372s. This indicates that the high overhead of HEG-Config stems primarily from its retrieval backend rather than from the method itself.

Motivated by this observation, we replace LightRAG with a naive RAG backend, referred to as HEG-Config-NR, as a simple cost-reduction attempt. As reported in Table 14, this reduces the training time from 6.28 hours to 1435.1s and the inference time from 15.6s to 9.13s. Meanwhile, as shown in Table 16, HEG-Config-NR attains a mean NPDI of 0.70, only slightly higher than HEG-Config's 0.68, with per-dataset results remaining consistently competitive. This suggests that HEG-Config's solving efficiency relies primarily on its hierarchical experience rather than on a sophisticated retrieval backend.