LEAP: LLM-EMPOWERED ADAPTIVE PORTFOLIO DISCOVERY FOR INSTANCE-AWARE MILP CONFIGURATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We study instance-aware solver configuration for mixed-integer linear programs (MILPs). Modern MILP solvers expose numerous tunable parameters, and different instances within the same problem family can benefit from different configurations. Portfolio-based approaches exploit this variation by discovering a set of complementary configurations in advance and selecting an appropriate configuration from this portfolio for each new instance. However, portfolio discovery requires expensive solver evaluations over large configuration spaces, while conventional configurators primarily rely on numerical performance feedback without directly leveraging solver parameter semantics or problem-family knowledge. To address this limitation, we propose **LEAP** (LLM-Empowered Adaptive Portfolio Discovery), which integrates semantic guidance into numerical search. An LLM *Initializer* proposes starting configurations using problem descriptions and parameter semantics, while an LLM *Refiner* uses portfolio performance and search feedback to adaptively focus or expand a working subspace within a fixed base configuration space, guiding an existing model-based configurator to search for configurations that improve the current portfolio. At deployment, a lightweight GNN selector, with no LLM calls or configuration search, selects the configuration with the lowest predicted solving cost for each incoming instance. Experiments on MILP benchmarks demonstrate solving-time reductions of up to 43.2% and 94.7% over the default SCIP and Gurobi configurations, respectively. Ablations demonstrate the benefits of knowledge-guided initialization and adaptive working-subspace refinement for discovering higher-quality configuration portfolios.

1 INTRODUCTION

Many real-world decision problems are naturally modeled as mixed-integer linear programs (MILPs). In applications such as hybrid model predictive control (Richards & How, 2005; Frick et al., 2019; Marcucci & Tedrake, 2020), dynamic vehicle routing (Dondo & Cerdá, 2006; Ozbaygin & Savelsbergh, 2019; Andersen et al., 2024), unit commitment (Morales-España et al., 2013; Gentile et al., 2017; He et al., 2024), and production planning (Wolsey, 1997; Cedillo-Robles et al., 2020; Dunke & Nickel, 2023), practitioners repeatedly solve related instances from the same problem class or modeling template. Although such instances share high-level formulations, they can differ in scale, constraint structure, sparsity patterns, and numerical data, making different solver configurations preferable for different instances.

Modern MILP solvers expose numerous configurable parameters controlling components such as presolving, heuristics, cutting planes, and branching strategies, whose settings can substantially affect computational efficiency (Li et al., 2025). Automatically selecting suitable parameter values is typically formulated as an *algorithm configuration* problem. Despite substantial progress in configuring MILP solvers (Li et al., 2023; Patel, 2024; Liu et al., 2024a), two important challenges remain. First, configuration preferences vary across instances. A single configuration optimized for average performance over a problem family may not fully exploit this variation. Portfolio-based approaches such as Hydra (Xu et al., 2011) address this challenge by constructing complementary configurations that perform well on different instances and selecting among them for new instances. Second, discovering such portfolios remains computationally expensive. Large parameter spaces and

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.

costly evaluations across multiple instances restrict the amount of exploration possible within a finite budget. Even model-based configurators primarily learn from numerical performance observations, without directly exploiting parameter semantics or problem-family knowledge.

This limitation motivates the following question: *How can problem-family knowledge and solver parameter semantics improve the discovery of complementary configurations for instance-aware MILP solving?* To address this question, we propose **LEAP** (LLM-Empowered Adaptive Portfolio Discovery), a configuration portfolio discovery method that integrates LLM-based semantic guidance into numerical search. LEAP introduces two complementary LLM roles into a Hydra-style iterative search procedure. The *Initializer* proposes starting configurations using problem descriptions and parameter semantics. The *Refiner* uses the current portfolio's performance and historical search feedback to adapt a working subspace within a fixed base configuration space, supplying the search domain for the next complementary-configuration search performed by an existing configurator such as SMAC. For each new instance, a trained GNN selects a configuration from the discovered portfolio with the lowest predicted solving cost, without additional LLM calls or configuration search.

Our main contributions are threefold: (i) We propose LEAP, an LLM-empowered adaptive portfolio discovery framework for instance-aware MILP configuration, incorporating initial candidate generation and working-subspace adaptation into iterative complementary-configuration search. (ii) We design two complementary LLM roles: an Initializer that proposes configurations using problem descriptions and parameter semantics, and a Refiner that uses portfolio performance and search feedback to adapt the working subspace within a fixed base configuration space. (iii) Extensive experiments with SCIP and Gurobi demonstrate solving-time reductions of up to 43.2% and 94.7% over their respective default configurations. Ablations demonstrate the contributions of both LLM roles to portfolio quality; cross-scale experiments further show that the discovered portfolios deliver solving-time gains on larger instances without additional configuration search or selector retraining.

2 Literature Review

Algorithm Configuration. Algorithm configuration methods can be broadly divided into instance-agnostic and instance-aware approaches. Instance-agnostic methods, including ParamILS, irace, GGA, and SMAC, seek a single configuration that performs well over a target instance set (Hutter et al., 2009; López-Ibáñez et al., 2016; Ansótegui et al., 2009; Hutter et al., 2011). However, heterogeneous MILP instances may favor different parameter settings, making a single configuration a compromise across diverse requirements. Instance-aware approaches adapt configurations to individual instances or instance groups. ISAC clusters instances using hand-crafted features and tunes one configuration per cluster (Kadioglu et al., 2010), while Hydra-MIP iteratively builds a portfolio of complementary configurations and learns an instance-level selector (Xu et al., 2011). Although these approaches exploit instance heterogeneity, portfolio construction requires repeated configuration search and solver evaluations. Under limited budgets, exploring large parameter spaces remains costly, particularly when problem knowledge and parameter semantics are not explicitly incorporated into the search. Moreover, hand-crafted features may provide an incomplete representation of MILP structure, motivating learned representations for configuration selection.

Learning-based MILP Solver Configuration. Learning-based approaches support instance-dependent configuration through candidate selection or direct parameter prediction. For candidate selection, GNN-based methods predict configuration performance from bipartite graph representations of MILP instances (Valentin et al., 2022). DGCAC learns graph embeddings through an auto-encoder, clusters instances, and tunes a configuration for each group (Song et al., 2023). Other approaches transfer configurations from nearby instances in a learned metric space (Hosny & Reda, 2024) or select settings for specific components such as separators (Li et al., 2023). These methods improve instance–configuration matching, but their performance also depends on the quality and diversity of the configurations available for selection.

Direct prediction instead maps instance representations to solver decisions. L2P-MIP predicts instance-specific presolving settings (Liu et al., 2024a), while multi-task representation learning supports solver-configuration prediction through shared MILP representations (Cai et al., 2025). Unlike portfolio-based selection, these methods learn instance-conditioned solver decisions directly

rather than selecting only from an explicitly constructed portfolio. However, the predicted decisions need not correspond to complete configurations that were explicitly evaluated during offline search.

LLM-enhanced Optimization. LLMs provide opportunities to incorporate semantic knowledge into optimization with expensive black-box evaluations. LLAMBO explores LLM-based initialization, surrogate modeling, and candidate sampling within Bayesian optimization (Liu et al., 2024b), while SLLMBO uses evaluation feedback to adapt hyperparameter search (Mahammadli & Ertekin, 2024). For MILP solver configuration, Lawless et al. (2025) use problem descriptions and separator information to generate candidate separator configurations, which are subsequently aggregated and selected. This work demonstrates the value of semantic guidance for cold-start configuration, while focusing on separator settings and LLM-generated candidates. It leaves room to investigate how semantic reasoning can guide iterative numerical search across multiple solver components.

LEAP combines LLM-guided initialization with SMAC-based configuration search (Hutter et al., 2011; Lindauer et al., 2022). Portfolio performance and search history guide adaptation of the working subspace within a fixed base configuration space, while a portfolio-aware objective guides the discovery of complementary configurations. A GNN then predicts configuration costs for instance-level selection from the empirically evaluated portfolio.

3 Adaptive Portfolio Discovery Framework

In this section, we introduce the MILP formulation and present an adaptive portfolio discovery framework under a limited search budget.

Problem Setup. An MILP problem takes the form

$$\min_{\boldsymbol{z}\in\mathbb{R}^n} \ \boldsymbol{c}^\top \boldsymbol{z} \quad \text{s.t.} \quad \boldsymbol{A}\boldsymbol{z} \circ \boldsymbol{b}, \quad \boldsymbol{l} \le \boldsymbol{z} \le \boldsymbol{u}, \quad z_j \in \mathbb{Z}\ \forall j \in \mathbb{I}. \tag{1}$$

where $\boldsymbol{A} \in \mathbb{Q}^{m\times n}$, $\boldsymbol{c} \in \mathbb{Q}^n$, and $\boldsymbol{b} \in \mathbb{Q}^m$ denote the constraint matrix, objective coefficients, and constraint right-hand sides, respectively. The variable bounds satisfy $l_j \in \mathbb{Q} \cup \{-\infty\}$ and $u_j \in \mathbb{Q}\cup\{+\infty\}$. The relation $\circ$ applies row-wise, with $\circ_i \in \{\le, =, \ge\}$ for $i = 1, \dots, m$. The index set $\mathbb{I} \subseteq \{1, \dots, n\}$ specifies the integer variables.

We consider heterogeneous MILP instances from a specific problem family, with instance space $\mathcal{X}$. These instances share a formulation template but differ in their input data, such as $\boldsymbol{c}$, $\boldsymbol{b}$, and $\boldsymbol{A}$, and problem size (m, n), resulting in different solving difficulties and configuration requirements.

For each of M selected solver parameters, we specify a finite set of admissible values $\mathcal{S}_i$. The fixed base configuration space is their Cartesian product, $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_M$. A complete configuration is denoted by $s \in \mathcal{S}$, with $[s]_i \in \mathcal{S}_i$ specifying its value for parameter i; all remaining solver parameters are held fixed. Let $c(s, x)$ denote the solving cost of applying configuration s to instance $x \in \mathcal{X}$. We focus on solving time, with time limits and timeout handling specified in the experimental setup.

Adaptive Portfolio Discovery. Instance heterogeneity motivates discovering a small portfolio $\mathcal{P} \subseteq \mathcal{S}$ whose configurations complement one another across a discovery dataset $\mathcal{D}_{\text{search}}$. Let $c(\mathcal{P}, x) = \min_{s'\in\mathcal{P}} c(s', x)$ denote the best cost attained by the portfolio on instance x. Following the portfolio-aware search mechanism of Hydra (Xu et al., 2011), discovery starts from initial candidates and iteratively searches for configurations that improve the current portfolio. Step 1 is performed once, followed by Steps 2–4 for R iterations.

Step 1: Portfolio Initialization. Given initial candidates $s_1^{(0)}, \dots, s_{N_{\text{init}}}^{(0)} \in \mathcal{S}$, we start with the default configuration and evaluate each candidate on $\mathcal{D}_{\text{search}}$, sequentially admitting those that reduce the current portfolio cost. This yields the initial portfolio $\mathcal{P}^{(0)}$ and evaluation history $\mathcal{H}^{(0)}$.

Step 2: Working-Subspace Specification. At iteration r, the current portfolio $\mathcal{P}^{(r-1)}$, search history $\mathcal{H}^{(r-1)}$, and previous working subspace $\mathcal{S}^{(r-1)}$ inform the choice of the next working subspace $\varnothing \neq \mathcal{S}^{(r)} \subseteq \mathcal{S}$, with $\mathcal{S}^{(0)} = \mathcal{S}$. The base space remains fixed, while the working subspace may be restricted or expanded by restoring previously excluded values.

Step 3: Complementary Configuration Search. Within $\mathcal{S}^{(r)}$, an algorithm configurator searches under a given budget to minimize $\sum_{x\in\mathcal{D}_{\text{search}}} c_{\text{eval}}(s, x \mid \mathcal{P}^{(r-1)})$ and returns a candidate $s^{(r)}$, where

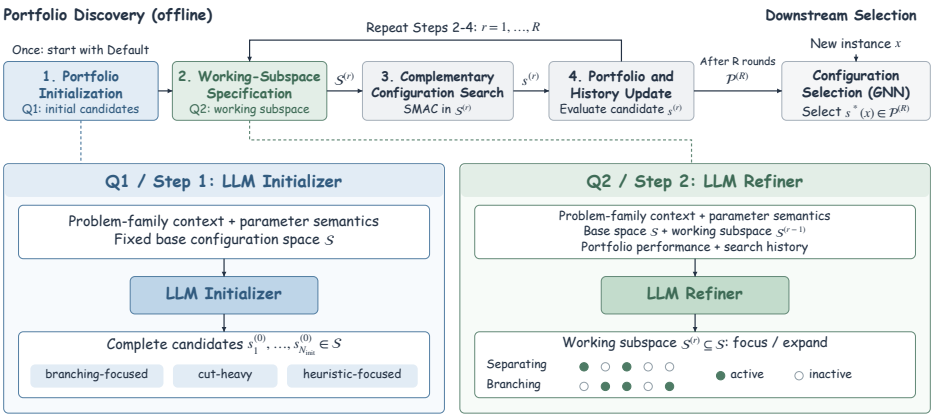

Figure 1: Overview of LEAP. The Initializer proposes candidates for Step 1, while the Refiner adapts the working subspace within the fixed base configuration space $\mathcal{S}$ in Step 2 for subsequent SMAC search. Dashed links indicate expanded views of these two interfaces.

$c_{\text{eval}}(s, x \mid \mathcal{P}) = \min\{c(s, x), c(\mathcal{P}, x)\}$. Thus, a candidate only needs to improve the current portfolio on a subset of instances, rather than be the best standalone configuration on average.

Step 4: Portfolio and History Update. We evaluate $s^{(r)}$ on $\mathcal{D}_{\text{search}}$ and admit it if $c(s^{(r)}, x) < c(\mathcal{P}^{(r-1)}, x)$ for at least one instance x; otherwise, the portfolio remains unchanged. This yields $\mathcal{P}^{(r)}$. In either case, the history $\mathcal{H}^{(r)}$ is updated to inform the next iteration. Initialization uses the same admission rule. After all iterations, redundant configurations are removed sequentially without changing the per-instance best costs on $\mathcal{D}_{\text{search}}$.

This framework leaves two key design questions: **(Q1)** How can problem structure and solver parameter semantics guide the generation of initial configurations with complementary potential? **(Q2)** How can portfolio performance and search history guide adaptation of the working subspace within the fixed base configuration space to discover complementary configurations more effectively?

4 LLM-Empowered Portfolio Discovery

LEAP combines problem-family knowledge and the semantics of solver parameters with empirical feedback to guide configuration portfolio discovery. As illustrated in Figure 1, LEAP addresses the two design questions posed in Section 3 using an LLM Initializer and an LLM Refiner. The Initializer proposes initial configurations with different strategic emphases. The Refiner uses portfolio performance and search history to adapt the working subspace within the fixed base configuration space; an algorithm configurator then searches this subspace for complementary configurations. We describe these two mechanisms below, followed by the downstream deployment of the discovered portfolio. The complete discovery algorithm is provided in Appendix C.

4.1 Configuration Portfolio Initialization

The Initializer constructs a diverse set of candidate configurations rather than committing to a single configuration shared by all instances. Its prompt contains three types of information.

Problem-family context. The prompt provides a natural-language description of the problem family, its MILP formulation, and its salient structural characteristics. It also identifies which data components vary across instances. This information contextualizes the configuration proposals without assuming that a particular structural property necessarily favors a specific parameter setting.

Parameter semantics and configuration space. We describe each parameter's function and trade-offs and enumerate its base admissible set $\mathcal{S}_i$. To instantiate a proposed strategy, the LLM selects one

admissible value for every tunable parameter from its corresponding set $\mathcal{S}_i$. The resulting combination forms a complete configuration $s \in \mathcal{S}$.

Output requirements. The LLM is instructed to propose N_{init} complete configurations. Each configuration must assign an admissible value to every tunable parameter and be accompanied by a brief description of its intended strategy and rationale. The candidates are requested to represent distinct strategic emphases, such as aggressive cut generation, primal-heuristic search, branching-focused search, or conservative overhead control.

The resulting candidates $s_1^{(0)}, \ldots, s_{N_{\text{init}}}^{(0)}$ are first checked for completeness and admissibility. They are then evaluated sequentially on $\mathcal{D}_{\text{search}}$ and admitted according to the portfolio-update rule in Section 3, yielding the initial portfolio $\mathcal{P}^{(0)}$ and evaluation history $\mathcal{H}^{(0)}$. The accompanying rationales describe the intended strategies, whereas admission depends only on measured solver performance. The box below presents a condensed excerpt from a recorded initialization prompt and response; complete initialization and refinement examples are provided in Appendix D.

Example Prompt & Response Snippet (Initialization):
Prompt Context: "...Load balancing with binary job assignments, capacity constraints, and a dense assignment structure. Propose 10 diverse SCIP configurations..."
LLM Response: **Strategy 3:** `branching-focused`
Params: `branching/scorefunc=q,`
`separating/clique/freq=10, heuristics/rins/freq=10, ...`
Rationale: Optimize branching decisions for the dense assignment matrix.
Other strategies: `cut-heavy, heuristic-focused, ...`

4.2 Adaptive Working-Subspace Refinement

As the portfolio develops, subsequent search seeks configurations that improve its existing coverage. LEAP therefore adapts the working subspace using accumulated feedback. At round r, the working subspace is $\mathcal{S}^{(r)} = \prod_{i=1}^{M} \mathcal{S}_i^{(r)}$, where $\varnothing \neq \mathcal{S}_i^{(r)} \subseteq \mathcal{S}_i$. The base configuration space $\mathcal{S}$ remains fixed, while refinement changes which admissible values are available to the configurator. Each refinement prompt contains three elements.

Problem and search-space context. The prompt includes the problem description and parameter semantics and provides, for each parameter i, its base admissible set $\mathcal{S}_i$, current working set $\mathcal{S}_i^{(r-1)}$, and previous refinement actions.

Portfolio and search feedback. The LLM receives the current portfolio $\mathcal{P}^{(r-1)}$, history $\mathcal{H}^{(r-1)}$, and recent search statistics, including observed configurations, their performance relative to the portfolio, and parameter-value summaries. These observations inform exploration but are not treated as causal estimates of individual parameter effects.

Output requirements. For each proposed change to parameter i, the LLM must specify an operation, an updated working set $\mathcal{S}_i^{(r)}$, and a rationale for the change. A `focus` action replaces the current working set with a nonempty strict subset, whereas an `expand` action retains the current values and restores additional values from the base admissible set $\mathcal{S}_i$. Parameters not targeted by an action remain unchanged.

Example Prompt & Response Snippet (Refinement):
Prompt Context: "...Current `mcf/freq` values: $\{0, 5\}$; mean relative costs for both are near 1. Base choices: $\{-1, 0, 1, 5, 10, 20\}$. Propose a working-subspace update..."
LLM Response: **Operation:** `expand`
Update: `mcf/freq`: $\{0, 5\} \rightarrow \{-1, 0, 5, 10\}$
Rationale: Weak, mixed gains motivate broader exploration within the base space.

The proposed actions are validated for parameter names, admissibility, nonempty value sets, and the required subset or superset relationships. Invalid outputs leave the current working subspace unchanged.

An algorithm configurator, instantiated as SMAC in this work, then searches $\mathcal{S}^{(r)}$ using the portfolio-aware objective defined in Section 3. Solver evaluations of the returned candidate update $\mathcal{P}^{(r)}$ and $\mathcal{H}^{(r)}$, providing feedback for the next refinement round.

4.3 Downstream Configuration Selection

To use the discovered portfolio on unseen instances, we train a GNN-based configuration selector. MILP instances are represented as variable–constraint bipartite graphs (Gasse et al., 2019), using the same feature set as BenLOC (Li et al., 2025). For a final portfolio of size $K = |\mathcal{P}^{(R)}|$, the predictor $f_\theta : \mathcal{X} \to \mathbb{R}^K$ outputs normalized solving times relative to Default, with $f_\theta^{(k)}(x)$ corresponding to configuration s_k.

We construct normalized targets from precomputed solving times on $\mathcal{D}_{\text{train}}$ and train the predictor using Smooth L1 loss. Normalization, architecture, and training details are provided in Appendix A.2. For an unseen instance x, the selector chooses $s^*(x) = \arg\min_{s_k \in \mathcal{P}^{(R)}} f_\theta^{(k)}(x)$, the configuration with the lowest predicted value.

5 Experimental Results

In this section, we evaluate LEAP against default solver configurations and representative baselines. Our experiments address the following research questions: (**RQ1**) *Overall performance.* How does LEAP perform relative to the default SCIP and Gurobi configurations and representative baselines? (**RQ2**) *Effectiveness of LLM-empowered portfolio discovery.* How do portfolio initialization and working-subspace refinement contribute to the quality of the discovered portfolios? (**RQ3**) *Cross-scale generalization.* How well do the discovered portfolios and trained selectors transfer, without adaptation, across planning horizons and problem scales? (**RQ4**) *Scaling with the discovery budget.* How does solving performance scale with the discovery budget?

Benchmarks and Experiment Setup. We evaluate LEAP on 12 diverse datasets from the Re-optimization (Bolusani et al., 2024) and BenLOC (Li et al., 2025) benchmarks. The 10 Re-optimization datasets are categorized into *easy* (single varying component) and *hard* (multiple components, large scale). To assess structural diversity, we include `load_balancing` (`load`) and `nn_verification` (`nn`) from BenLOC. All experiments use SCIP 9.0.2 and Gurobi 12.0.3 with three random seeds. We enforce a 10-hour total SMAC search budget for all search-based methods. Portfolio-based approaches use $r = 5$ iterations, with 2 hours of search per iteration using SMAC (Lindauer et al., 2022) as the algorithm configurator; variants with LLM-based initialization use $|\mathcal{P}_0| = 10$. Discovery and selector training are offline; online selection requires no LLM calls or configuration search. For example, on SCIP `load`, selection including graph construction takes 0.24 s per instance versus 217.7 s for solving (Appendix A.3). Detailed descriptions of experiment setup and benchmarks can be found in Appendix A.1 and Appendix B, respectively.

Evaluation Metrics. Following our formulation, we measure the performance cost $c(s, x)$ as solving time. Let $\mathcal{D}_{\text{test}}$ denote the test set of N instances. Our primary metric is the *average solving time*: $T = \frac{1}{N} \sum_{x \in \mathcal{D}_{\text{test}}} t(x)$, where $t(x)$ is the time taken to solve instance x. We report the improvement over the default solver, Imp. $= \frac{T_{\text{def}} - T}{T_{\text{def}}} \times 100\%$, where T_{def} is its average solving time. Since performance depends on both portfolio quality and configuration selection, we also report *per-instance best (PI)* performance to assess portfolio quality independently of the selector. PI assumes a hypothetical perfect selector that always chooses the best configuration from the portfolio for each instance. Instances are solved to optimality, except for `load`, which uses a relative optimality gap of 0.05, with unsolved instances assigned the 500-second time limit.

Baselines. To evaluate overall performance, we compare **LEAP** against: (i) **Default**: SCIP and Gurobi with their respective default configurations. (ii) **DGCAC**: A state-of-the-art instance-specific method that clusters instances using unsupervised GNN embeddings (Song et al., 2023), representing a clustering-based approach to configuration portfolio construction. (iii) **LLM-Cold-Start**, which uses the configurations from the LLM initializer without SMAC optimization (Lawless et al., 2025), selects the best of $|\mathcal{P}_0| = 10$ configurations on the validation set, and applies that single configuration

Table 1: **(RQ1)** SCIP results: mean solving time (s) and improvement over Default (Imp., %). Both denotes Init and Refine. Cold-Start refers to LLM-cold-start.

	Default	DGCAC		Global-SMAC		Cold-Start		w/o Both		w/o Refine		w/o Init		LEAP	
Dataset	Time ↓	Time ↓	Imp. ↑	Time ↓	Imp. ↑	Time ↓	Imp. ↑	Time ↓	Imp. ↑	Time ↓	Imp. ↑	Time ↓	Imp. ↑	Time ↓	Imp. ↑
Easy datasets															
bnd_2	248.7	259.9	-4.5	201.5	19.0	270.4	-8.7	212.9	14.4	218.1	12.3	228.0	8.3	**195.6**	**21.4**
bnd_3	420.2	408.1	2.9	**379.9**	**9.6**	408.6	2.8	390.9	7.0	383.2	8.8	403.1	4.1	394.3	6.2
rhs_2	92.1	61.7	33.0	66.9	27.4	54.3	41.0	60.6	34.2	53.3	42.1	60.4	34.4	**52.2**	**43.2**
rhs_4	89.5	84.0	6.1	67.5	24.6	99.1	-10.7	65.1	27.3	62.9	29.7	62.8	29.8	**59.5**	**33.6**
obj_1	465.8	436.6	6.3	415.7	10.8	463.7	0.5	412.1	11.5	414.4	11.0	**392.1**	**15.8**	399.6	14.2
Hard datasets															
rhs_3	325.4	327.1	-0.5	317.4	2.5	299.5	8.0	313.1	3.8	297.4	8.6	321.7	1.1	**290.7**	**10.7**
obj_2	392.3	398.4	-1.6	364.5	7.1	394.8	-0.6	383.7	2.2	365.5	6.8	387.0	1.4	**364.3**	**7.1**
mix_1	191.0	154.2	19.3	146.3	23.4	125.9	34.1	155.7	18.5	150.9	21.0	156.3	18.2	**125.3**	**34.3**
mix_2	285.7	264.0	7.6	**259.2**	**9.3**	273.9	4.1	285.8	0.0	268.1	6.2	297.5	-4.1	264.4	7.5
mix_3	98.0	91.4	6.7	95.3	2.8	134.5	-37.2	111.8	-14.1	98.8	-0.8	99.6	-1.6	**86.0**	**12.1**
BenLOC datasets															
nn	77.4	65.6	15.2	72.3	6.6	81.9	-5.8	58.9	23.9	55.8	27.9	60.5	21.8	**52.9**	**31.8**
load	250.7	348.6	-39.1	241.9	3.5	243.9	2.7	252.3	-0.6	241.1	3.8	235.5	6.1	**217.7**	**13.2**

Table 2: **(RQ1)** Gurobi results: mean solving time (s) and improvement over Default (Imp., %). Both denotes Init and Refine. Cold-Start refers to LLM-cold-start.

	Default	DGCAC		Global-SMAC		Cold-Start		w/o Both		w/o Refine		w/o Init		LEAP	
Dataset	Time ↓	Time ↓	Imp. ↑	Time ↓	Imp. ↑	Time ↓	Imp. ↑	Time ↓	Imp. ↑	Time ↓	Imp. ↑	Time ↓	Imp. ↑	Time ↓	Imp. ↑
Easy datasets															
bnd_2	271.5	251.8	7.3	**249.6**	**8.1**	282.8	-4.2	270.2	0.5	271.5	0.0	274.3	-1.0	269.3	0.8
bnd_3	277.0	273.8	1.2	263.1	5.0	273.3	1.3	278.4	-0.5	261.2	5.7	277.3	-0.1	**258.5**	**6.7**
rhs_2	14.2	10.3	27.5	11.1	21.8	7.5	47.2	6.6	53.5	6.4	54.9	6.2	56.3	**6.1**	**57.2**
rhs_4	15.1	9.7	35.8	12.6	16.6	7.5	50.3	6.2	58.9	6.0	60.3	6.2	58.9	**5.2**	**65.5**
obj_1	109.4	**98.9**	**9.6**	105.3	3.7	99.0	9.5	109.2	0.2	108.2	1.1	114.7	-4.8	103.7	5.2
Hard datasets															
rhs_3	23.7	23.0	3.0	25.8	-8.9	23.5	0.8	25.1	-5.9	34.8	-46.8	31.5	-32.9	**22.9**	**3.7**
obj_2	27.7	28.3	-2.2	25.7	7.2	13.1	52.7	4.1	85.2	5.3	80.9	**1.2**	**95.7**	1.5	94.5
mix_1	12.6	13.6	-7.9	12.2	3.2	13.4	-6.3	12.3	2.4	13.4	-6.3	12.6	0.0	**12.0**	**4.2**
mix_2	59.0	47.8	19.0	45.7	22.5	23.3	60.5	26.6	54.9	24.7	58.1	40.4	31.5	**22.5**	**62.1**
mix_3	9.5	8.8	7.4	8.6	9.5	11.7	-23.2	13.1	-37.9	7.7	18.9	15.0	-57.9	**6.6**	**30.1**
BenLOC datasets															
nn	6.2	5.1	17.7	5.2	16.1	5.6	9.7	5.1	17.7	4.9	21.0	5.9	4.8	**4.6**	**26.0**
load	14.9	3.5	76.5	1.3	91.3	2.1	85.9	0.9	94.0	0.8	94.6	0.9	94.0	**0.8**	**94.7**

to all test instances, evaluating LLM-generated configurations without subsequent numerical search. (iv) **Global-SMAC**, which uses the full 10-hour SMAC search budget to find a single configuration with the best average performance on $\mathcal{D}_{\text{search}}$. We also conduct three ablations to isolate the gains of each component of LEAP: (a) **LEAP w/o Init & Refine** (w/o Both), which performs standard iterative portfolio discovery from Default without either LLM-based initialization or refinement, following Hydra (Xu et al., 2011); (b) **LEAP w/o Refine**, which employs LLM-based initialization but performs standard iterative portfolio discovery without the LLM Refiner; and (c) **LEAP w/o Init**, which starts from Default and retains the LLM Refiner. LEAP and its three ablations use the same GNN-based selection protocol. Further details regarding the baselines can be found in Appendix A.4.

5.1 Experiment Results

(RQ1) Overall performance. LEAP improves solving performance across both solvers. Tables 1 and 2 show that LEAP achieves the lowest mean solving time on 9 of 12 benchmarks for SCIP and Gurobi, respectively, and outperforms Default on all 24 solver–benchmark pairs. Improvements over Default reach 43.2% on SCIP `rhs_2` and 94.7% on Gurobi `load`. LEAP also outperforms DGCAC on 21 of the 24 solver–benchmark pairs, demonstrating gains not only over solver defaults but also over an existing instance-specific configuration method. These results span different solvers, problem structures, and forms of instance variation, showing that LEAP-discovered portfolios translate into broad practical solving-time gains through downstream selection. The evaluated benchmarks include challenging settings where Default solves only 39.7–84.0% of test instances within 500 seconds. LEAP improves solve rates on all seven such SCIP benchmarks (Appendix E.2).

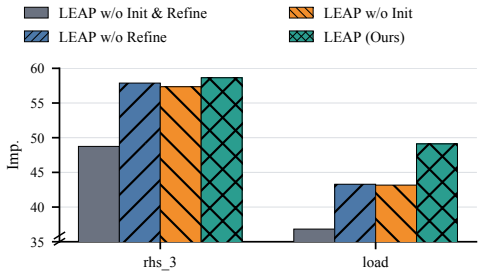

Figure 2: **RQ2: Final portfolio quality.** PI improvement over Default on SCIP $\mathcal{D}_{\text{search}}$ after five discovery iterations. Higher is better.

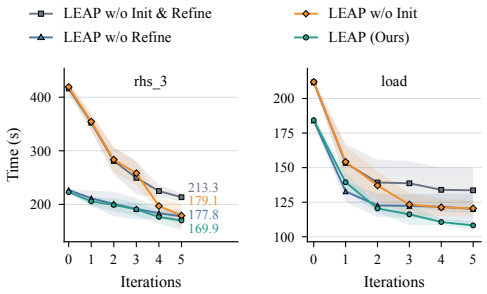

Figure 3: **RQ2: Discovery trajectories.** Mean PI solving time on SCIP $\mathcal{D}_{\text{search}}$ across discovery iterations. Lower is better.

Complementary-configuration search builds on effective LLM-generated starting configurations. By selecting just one of ten LLM-proposed candidates on the validation set, LLM-Cold-Start improves over Default by 47.2% on Gurobi `rhs_2` and 60.5% on `mix_2`. These results support the value of problem descriptions and solver parameter semantics for initialization. However, generating candidates and selecting a single configuration does not fully exploit their different strengths across instances. By combining these suggestions with numerical search for complementary configurations, LEAP outperforms LLM-Cold-Start on 23 of 24 solver–benchmark pairs, reducing solving time from 99.1 to 59.5 seconds on SCIP `rhs_4` and from 13.1 to 1.5 seconds on Gurobi `obj_2`. LLM-Cold-Start slightly outperforms LEAP on Gurobi `obj_1`, showing that effective initial configurations can also support competitive static deployment on some benchmarks. Since this comparison involves differences in both subsequent discovery and deployment, RQ2 further examines the contributions of the two LLM components to portfolio quality.

LEAP outperforms unguided portfolio search and Global-SMAC overall. Full LEAP achieves lower mean solving times than LEAP w/o Init & Refine on 11 of 12 SCIP benchmarks and all 12 Gurobi benchmarks. LEAP and LEAP w/o Init & Refine share the same portfolio-update mechanism, 10-hour SMAC search budget, and downstream GNN training protocol, supporting the combined value of LLM initialization and refinement for portfolio discovery. Global-SMAC devotes the entire 10-hour search budget to optimizing a single configuration, whereas LEAP distributes the same search budget across iterative searches for complementary configurations. Under this equal search budget, LEAP outperforms Global-SMAC on 21 of 24 solver–benchmark pairs, reducing solving time from 25.7 to 1.5 seconds on Gurobi `obj_2` and from 12.6 to 5.2 seconds on `rhs_4`. These results show that discovering and selecting among complementary configurations can be more effective than concentrating the search budget on a single global configuration.

(RQ2) Effectiveness of LLM-empowered portfolio discovery. Combining both LLM roles yields the best portfolios on both benchmarks. Figure 2 compares the final PI performance of the four discovery variants on SCIP `rhs_3` and `load`, directly assessing the quality of the discovered portfolios on $\mathcal{D}_{\text{search}}$. On `rhs_3`, final PI costs are 213.3, 177.8, 179.1, and 169.9 seconds for LEAP w/o Init & Refine, LEAP w/o Refine, LEAP w/o Init, and full LEAP, respectively; `load` shows the same relative ordering. Each component improves portfolio quality individually, while combining them yields further gains.

The initializer provides a stronger starting portfolio. In Figure 3, LEAP w/o Refine and full LEAP already outperform the default-initialized variants at iteration zero. By using problem descriptions and parameter semantics to propose candidates, the Initializer gives discovery a stronger starting portfolio, reducing its reliance on budget-limited numerical search for initial improvements.

The refiner improves portfolios beyond initialization alone. Under the same number of iterations and SMAC search budget, full LEAP achieves lower final PI costs than LEAP w/o Refine on both benchmarks. Without the initializer, LEAP w/o Init also outperforms LEAP w/o Init & Refine, showing that refinement benefits are not limited to LLM-initialized portfolios. Figure 3 further shows gains in later iterations, particularly on `load`, where full LEAP continues to improve as gains without refinement become smaller. By adapting the working subspace using portfolio performance and

search feedback, the Refiner guides where SMAC searches for further complementary configurations. Test-set PI comparisons across all 12 benchmarks and both solvers are reported in Appendix E.1.

(RQ3) Cross-scale generalization. We examine generalization in two respects: whether discovered portfolios remain effective on larger instances, and whether trained selectors can exploit these portfolios. We therefore transfer the portfolios and GNNs obtained on the six-week `mix_2` benchmark with Gurobi to 13- and 21-week instances, reporting PI and GNN performance separately. Detailed settings are provided in Appendix A.5.

LEAP transfers to larger instances at both the portfolio and selector levels. As shown in Table 3, LEAP's PI solving times are 26.1% and 17.7% below Default on the 13- and 21-week groups, respectively, whereas the portfolios discovered by LEAP w/o Init & Refine perform close to Default. These results show that the configurations discovered with LLM-guided initialization and refinement remain effective beyond the source problem distribution. Using the GNN selector obtained on the six-week `mix_2` benchmark, LEAP also reduces solving time by 7.1% and 5.4%, without additional LLM calls, configuration search, or GNN training. Thus, the discovered portfolios can be deployed on larger instances with immediate solving-time benefits. The remaining gap between PI and GNN performance indicates that some portfolio gains are not fully realized, leaving downstream selection as a bottleneck to exploiting the available configurations.

Table 3: **RQ3: Cross-scale generalization.** Gurobi time (s), std. dev. below.

Method	Default	PI	GNN
	13 weeks		
LEAP	181.6	**134.2**	**168.7**
	(4.8)	(3.2)	(29.1)
w/o Init & Refine	181.4	178.9	262.4
	(6.0)	(4.7)	(22.7)
	21 weeks		
LEAP	376.8	**310.2**	**356.4**
	(5.6)	(17.0)	(21.8)
w/o Init & Refine	379.5	379.5	440.3
	(4.6)	(4.4)	(26.7)

(RQ4) Scaling with the discovery budget. Additional discovery yields further performance gains.

We compare five, seven, and nine discovery iterations on SCIP `load` and `rhs_3`, with two hours of SMAC search per iteration and a separately trained GNN for each portfolio (Appendix A.5). As shown in Table 4, improvements over Default increase from 13.2% to 24.4% on `load` and from 10.7% to 19.9% on `rhs_3` when extending discovery from five to nine iterations. Gains continue beyond seven iterations: improvements rise from 19.4% to 24.4% on `load` and from 13.7% to 19.9% on `rhs_3`. These results show that additional discovery can further improve downstream solving performance beyond the standard five-iteration budget.

Table 4: **RQ4: Discovery iterations.** SCIP time (s), improvement (%) below. Each portfolio uses a separately trained GNN.

Dataset	Default	5 iter. 10 h	7 iter. 14 h	9 iter. 18 h
`load`	250.7	217.7	202.1	**189.7**
		13.2%	19.4%	24.4%
`rhs_3`	325.4	290.6	280.9	**260.8**
		10.7%	13.7%	19.9%

6 Conclusions

In this work, we propose LEAP, an LLM-empowered adaptive portfolio discovery framework for MILP solver configuration. LEAP uses problem descriptions and parameter semantics to generate initial configurations and iteratively refines the search subspace within a fixed base configuration space based on portfolio performance and search feedback, guiding numerical search toward complementary configurations. Experiments with SCIP and Gurobi show that the discovered portfolios deliver broad solving-time improvements through downstream selection; ablation studies further demonstrate the contributions of initialization and refinement to portfolio quality. Cross-scale experiments show that source-discovered portfolios retain their performance advantages on larger instances, while additional discovery iterations and search budget yield further gains on the evaluated benchmarks. These results highlight the value of problem knowledge and parameter semantics in complementing numerical performance feedback for solver configuration portfolio discovery. We discuss the limitations of our work in Appendix F.

AI USE STATEMENT

Generative AI tools were used during this research and manuscript preparation to improve the clarity and organization of the manuscript and polish the language. Separately, LEAP incorporates an LLM for configuration initialization and search-space refinement. This use is part of the proposed method and is documented in the main text and appendices. All experimental results and scientific claims were checked by the authors, and all AI-assisted text was reviewed and revised before submission. The authors take full responsibility for the final content of the paper, including all text, claims, code, and artifacts produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

The main paper describes the task formulation, proposed method, benchmarks, baselines, and overall experimental setup. Appendix A provides detailed configuration spaces, implementation settings, computational costs, and evaluation protocols, while Appendix B describes the datasets and splits. Appendices C and D provide the complete discovery procedure and LLM prompts needed to reproduce the reported results.

REFERENCES

Tiria Andersen, Shaun Belward, Mangalam Sankupellay, Trina Myers, and Carla Chen. Reoptimisation strategies for dynamic vehicle routing problems with proximity-dependent nodes. *Top*, 32(1): 1–21, 2024.

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.

Suresh Bolusani, Mathieu Besançon, Ambros Gleixner, Timo Berthold, Claudia D'ambrosio, Gonzalo Muñoz, Joseph Paat, and Dimitri Thomopulos. The mip workshop 2023 computational competition on reoptimization. *Mathematical Programming Computation*, 16(2):255–266, 2024.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

Junyang Cai, Taoan Huang, and Bistra Dilkina. Multi-task representation learning for mixed integer linear programming. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 134–151. Springer, 2025.

Juan Antonio Cedillo-Robles, Neale R Smith, Rosa G González-Ramirez, Julio Alonso-Stocker, Joaquín Alonso-Stocker, and Ronald G Askin. A production planning milp optimization model for a manufacturing company. In *International Conference of Production Research–Americas*, pp. 85–96. Springer, 2020.

Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In *International symposium on automated technology for verification and analysis*, pp. 251–268. Springer, 2017.

Rodolfo Dondo and Jaime Cerdá. An milp framework for dynamic vehicle routing problems with time windows. *Latin American applied research*, 36(4):255–261, 2006.

Fabian Dunke and Stefan Nickel. Exact reoptimisation under gradual look-ahead for operational control in production and logistics. *International Journal of Systems Science: Operations & Logistics*, 10(1):2141590, 2023.

Damian Frick, Angelos Georghiou, Juan L Jerez, Alexander Domahidi, and Manfred Morari. Low-complexity method for hybrid mpc with local guarantees. *SIAM Journal on Control and Optimization*, 57(4):2328–2361, 2019.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

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.

Claudio Gentile, Germán Morales-Espana, and Andres Ramos. A tight mip formulation of the unit commitment problem with start-up and shut-down constraints. *EURO Journal on Computational Optimization*, 5(1):177–201, 2017.

Xuan He, Honglin Wen, Yufan Zhang, Yize Chen, and Danny HK Tsang. Fast unit commitment constraint screening with learning-based cost model. In *2024 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pp. 295–300. IEEE, 2024.

Abdelrahman Hosny and Sherief Reda. Automatic milp solver configuration by learning problem similarities. *Annals of Operations Research*, 339(1):909–936, 2024.

Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of artificial intelligence research*, 36:267–306, 2009.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pp. 507–523. Springer, 2011.

Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. Isac–instance-specific algorithm configuration. In *ECAI 2010*, pp. 751–756. Ios Press, 2010.

Connor Lawless, Yingxi Li, Anders Wikum, Madeleine Udell, and Ellen Vitercik. Llms for cold-start cutting plane separator configuration. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 51–69. Springer, 2025.

Hongpei Li, Ziyan He, Yufei Wang, Wenting Tu, Shanwen Pu, Qi Deng, and Dongdong Ge. Benloc: A benchmark for learning to configure mip optimizers. *arXiv preprint arXiv:2506.02752*, 2025.

Sirui Li, Wenbin Ouyang, Max Paulus, and Cathy Wu. Learning to configure separators in branch-and-cut. *Advances in Neural Information Processing Systems*, 36:60021–60034, 2023.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

Chang Liu, Zhichen Dong, Haobo Ma, Weilin Luo, Xijun Li, Bowen Pang, Jia Zeng, and Junchi Yan. L2p-mip: Learning to presolve for mixed integer programming. In *The Twelfth International Conference on Learning Representations*, 2024a.

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

Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

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

Tobia Marcucci and Russ Tedrake. Warm start of mixed-integer programs for model predictive control of hybrid systems. *IEEE Transactions on Automatic Control*, 66(6):2433–2448, 2020.

Germán Morales-España, Jesus M Latorre, and Andres Ramos. Tight and compact milp formulation for the thermal unit commitment problem. *IEEE Transactions on Power Systems*, 28(4):4897–4908, 2013.

Gizem Ozbaygin and Martin Savelsbergh. An iterative re-optimization framework for the dynamic vehicle routing problem with roaming delivery locations. *Transportation Research Part B: Methodological*, 128:207–235, 2019.

Krunal Kishor Patel. Progressively strengthening and tuning mip solvers for reoptimization. *Mathematical Programming Computation*, 16(2):267–295, 2024.

Arthur Richards and Jonathan How. Mixed-integer programming for control. In *Proceedings of the 2005, American Control Conference, 2005.*, pp. 2676–2683. IEEE, 2005.

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.

Romeo Valentin, Claudio Ferrari, Jérémy Scheurer, Andisheh Amrollahi, Chris Wendler, and Max B Paulus. Instance-wise algorithm configuration with graph neural networks. *arXiv preprint arXiv:2202.04910*, 2022.

Laurence A Wolsey. Mip modelling of changeovers in production planning and scheduling problems. *European Journal of Operational Research*, 99(1):154–165, 1997.

Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Hydra-mip: Automated algorithm configuration and selection for mixed integer programming. In *RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence (IJCAI)*, pp. 16–30, 2011.

A Experiment Setups

A.1 Experiment Details of LLM-Guided Configuration Search

Experiments use SCIP 9.0.2 and Gurobi 12.0.3 on a high-performance computing cluster equipped with Intel Xeon Platinum 8628 CPUs and NVIDIA A100 Tensor Core GPUs. The updated main experiments use three random seeds. The LLM components use `gpt-4o` through the OpenAI API.

The standard configuration-search budget is ten hours of SMAC per benchmark. LEAP first evaluates ten LLM-proposed candidates on $\mathcal{D}_{\text{search}}$, retaining configurations with positive marginal contributions. It then performs five discovery iterations, each with two hours of serial SMAC search. After each iteration, the returned configuration is evaluated on all discovery instances outside the SMAC budget; cached costs for existing configurations are reused. The next search objective uses $\min(c(s, x), c(\mathcal{P}, x))$, reflecting the updated portfolio. After construction, redundant configurations are removed if their removal leaves the per-instance portfolio costs unchanged. The final portfolio is then used for downstream selection.

Configuration search space. Modern MILP solvers like SCIP expose thousands of tunable parameters. Many of these parameters, such as execution frequencies or iteration limits, have unbounded or extremely large integer domains (e.g., spanning from -1 to ∞). Direct optimization over this raw, quasi-continuous space is computationally intractable and highly inefficient for black-box configurators like SMAC.

To ensure efficient exploration and meaningful LLM guidance, we carefully curate a bounded, discrete configuration space. For instance, SCIP frequency parameters typically follow a specific semantic structure: -1 disables the component, 0 applies it only at the root node, and $n > 0$ applies it at every n-th depth level of the search tree. Accordingly, we discretize these unbounded ranges into a small set of representative categorical choices (e.g., $\{-1, 0, 1, 5, 10, 20\}$).

Furthermore, unlike many prior learning-based approaches that restrict their focus to a single solver component (e.g., only tuning separators or only tuning heuristics), we explicitly define a *joint configuration space*. The fixed configuration spaces contain 25 SCIP parameters and 29 Gurobi parameters, listed in Tables 5 and 6, respectively. The SCIP space spans four core solver components:

- **Separating:** Frequencies of various cutting plane generators (e.g., clique, implied bounds, Gomory).
- **Branching:** Variable selection strategies and pseudo-cost weighting factors.
- **Heuristics:** Execution frequencies of primal heuristics (e.g., RINS, ALNS) to discover early incumbents.
- **Presolving:** Limits on presolve rounds and early abort factors.

This parameter space allows our framework to discover complex synergies between different solver components, such as compensating for weak presolving with aggressive root-node cuts and intensive primal heuristics.

A.2 Implementation Details of the GNN Selector

Bipartite Graph Representation. Following the canonical representation for MILP instances introduced by Gasse et al. (2019) and BenLOC Li et al. (2025), we encode each problem instance as a heterogeneous bipartite graph $G = (\mathcal{V}_c, \mathcal{V}_v, \mathcal{E})$. Here, $\mathcal{V}_c$ represents the constraint nodes, $\mathcal{V}_v$ represents the variable nodes, and the edges $\mathcal{E}$ correspond to the non-zero coefficients in the constraint matrix. To facilitate bidirectional message passing, we model $\mathcal{E}$ as two directed relation types: constraint-to-variable and variable-to-constraint. The specific node and edge features utilized in our framework are summarized in Table 7.

Heterogeneous Graph Attention Network (HGAT). To capture the complex structural dependencies within the MILP instances, we adopt a Heterogeneous Graph Attention Network (GATv2) Brody et al. (2021), building upon the architectural designs of recent MIP learning frameworks (e.g., BenLOC). The model applies attention-based message passing across the bipartite relations. Let $h_i^{(c)}$ and

Table 5: SCIP parameter space used in our experiments.

Parameter	Meaning (brief)	Candidate values	Default
Separating (cut-generation frequency)			
`separating/aggregation/freq`	Aggregation separator frequency	{-1, 0, 1, 5, 10, 20}	10
`separating/clique/freq`	Clique separator frequency	{-1, 0, 1, 5, 10, 20}	0
`separating/mcf/freq`	Mixed-integer flow separator frequency	{-1, 0, 1, 5, 10, 20}	0
`separating/impliedbounds/freq`	Implied-bound separator frequency	{-1, 0, 1, 5, 10, 20}	10
`separating/cmir/freq`	CMIR separator frequency	{-1, 0, 1, 5, 10, 20}	10
`separating/strongcg/freq`	Strong Chvátal–Gomory separator frequency	{-1, 0, 1, 5, 10, 20}	10
`separating/zerohalf/freq`	Zero-half separator frequency	{-1, 0, 1, 5, 10, 20}	10
`separating/gomory/freq`	Gomory separator frequency	{-1, 0, 1, 5, 10, 20}	10
`separating/cgmip/freq`	CgMIP separator frequency	{-1, 0, 1, 5, 10, 20}	-1
`separating/oddcycle/freq`	Odd-cycle separator frequency	{-1, 0, 1, 5, 10, 20}	-1
`separating/rapidlearning/freq`	Rapid-learning call frequency	{-1, 0, 1, 5, 10, 20}	5
Branching			
`branching/scorefunc`	Branching score function	{p, s, q}	p
`branching/scorefac`	Weight in branching-score computation	{0.0, 0.167, 0.5, 0.75, 1.0}	0.167
`branching/preferbinary`	Whether to prefer binary variables	{True, False}	False
`branching/lpgainnormalize`	LP-gain normalization scheme	{d, l, s}	s
`branching/delaypscostupdate`	Whether to delay pseudo-cost updates	{True, False}	True
Primal heuristics (execution frequency)			
`heuristics/trivial/freq`	Trivial heuristic frequency	{-1, 0, 1, 5, 10, 20}	0
`heuristics/rins/freq`	RINS heuristic frequency	{-1, 0, 1, 5, 10, 15, 20, 25}	15
`heuristics/proximity/freq`	Proximity heuristic frequency	{-1, 0, 1, 5, 10, 20}	-1
`heuristics/pscostdiving/freq`	Pseudo-cost diving frequency	{-1, 0, 1, 5, 10, 20}	10
`heuristics/randrounding/freq`	Randomized-rounding frequency	{-1, 0, 1, 5, 10, 20}	10
`heuristics/alns/freq`	ALNS heuristic frequency	{-1, 0, 1, 5, 10, 20}	20
Presolving			
`presolving/maxrounds`	Maximum number of presolve rounds	{-1, 0, 1, 5, 10, 20}	-1
`presolving/maxrestarts`	Maximum number of presolve restarts	{-1, 0, 1, 5, 10, 20}	-1
`presolving/abortfac`	Factor controlling early presolve termination	{0.0, 0.0008, 0.001, 0.01, 0.1}	0.0008

$h_j^{(v)}$ denote the embeddings of constraint i and variable j, respectively, and let e_{ij} denote the edge feature (the scaled matrix coefficient) on link (i, j). For a given attention head, the message from j to i is computed as:

$$e_{j\to i} \;=\; \mathrm{LeakyReLU}\Big(\mathbf{a}^\top\Big(\Theta_s\, h_j^{(s)} + \Theta_t\, h_i^{(t)} + \Theta_e\, e_{ji}\Big)\Big), \qquad \alpha_{j\to i} \;=\; \frac{\exp(e_{j\to i})}{\sum_{k\in\mathcal{N}(i)}\exp(e_{k\to i})}, \tag{2}$$

where (s, t) corresponds to either (c, v) or (v, c) depending on the message direction, $\mathcal{N}(i)$ denotes the neighborhood of node i, and $\Theta_\cdot, \mathbf{a}$ are learnable weight matrices.

Multiple attention heads are concatenated to produce a hidden dimension d_h. After each HGAT block, we apply GraphNorm and a residual connection with a LeakyReLU activation. The initial raw node features are projected to dimension d_h using a Fourier feature encoder and a linear layer. After L message-passing blocks, we apply global sum pooling separately over the variable nodes and the constraint nodes. These pooled representations are concatenated and passed through a Multi-Layer Perceptron (MLP) to output a prediction vector of dimension $K = |\mathcal{P}|$, corresponding to the number of configurations in the portfolio. For instance x_i and configuration s_k, the regression target is $r_{ik} = (t_{ik} - t_{i,\text{default}})/t_{i,\text{default}}$, where t_{ik} and $t_{i,\text{default}}$ are their precomputed solving times. We minimize Smooth L1 loss with $\beta = 1$, averaged over instances and configurations. This normalization preserves the configuration ranking within each instance; lower predicted values indicate better configurations.

A.3 Computational Cost and Deployment Overhead

We report computational costs on `load`, separating offline portfolio discovery, label generation, and selector training from online configuration selection. Configuration evaluations and label generation use 20 parallel single-thread CPU workers. SMAC search and GNN training are timed separately.

Offline portfolio discovery. The standard setting consists of five SMAC rounds with two hours per round, giving a total search budget of ten hours. Outside this budget, we evaluate Default and ten initial configurations on the discovery set, followed by one new configuration after each round

Table 6: Gurobi parameter space used in our experiments.

Parameter	Meaning (brief)	Candidate values	Default
	Cut generation		
`Cuts`	Global cut aggressiveness	{-1, 0, 1, 2, 3}	-1
`CliqueCuts`	Clique-cut generation level	{-1, 0, 1, 2}	-1
`CoverCuts`	Cover-cut generation level	{-1, 0, 1, 2}	-1
`FlowCoverCuts`	Flow-cover-cut generation level	{-1, 0, 1, 2}	-1
`MIRCuts`	MIR-cut generation level	{-1, 0, 1, 2}	-1
`StrongCGCuts`	Strong Chvátal–Gomory cut level	{-1, 0, 1, 2}	-1
`ZeroHalfCuts`	Zero-half-cut generation level	{-1, 0, 1, 2}	-1
`ImpliedCuts`	Implied-bound-cut generation level	{-1, 0, 1, 2}	-1
`InfProofCuts`	Infeasibility-proof-cut generation level	{-1, 0, 1, 2}	-1
`GomoryPasses`	Number of root Gomory-cut passes	{-1, 0, 1, 2, 5, 10, 20}	-1
`CutPasses`	Maximum number of cutting-plane passes	{-1, 0, 1, 2, 5, 10, 20}	-1
	Search focus and primal heuristics		
`MIPFocus`	Search emphasis (balanced, feasibility, optimality, or bound)	{0, 1, 2, 3}	0
`Heuristics`	Fraction of effort devoted to primal heuristics	{0.0, 0.01, 0.05, 0.1, 0.25, 0.5, 1.0}	0.05
`RINS`	RINS heuristic frequency in nodes	{-1, 0, 1, 5, 10, 25, 50}	-1
`PumpPasses`	Number of feasibility-pump passes	{-1, 0, 1, 2, 5, 10, 20}	-1
`MinRelNodes`	Node budget for the minimum-relaxation heuristic	{-1, 0, 100, 500, 2000, 5000}	-1
`ZeroObjNodes`	Node budget for the zero-objective heuristic	{-1, 0, 100, 500, 2000, 5000}	-1
`SubMIPNodes`	Node budget for sub-MIP heuristics	{0, 50, 500, 2000}	500
	Branching and degeneracy handling		
`VarBranch`	Branching-variable selection strategy	{-1, 0, 1, 2, 3}	-1
`BranchDir`	Preferred branch direction	{-1, 0, 1}	0
`DegenMoves`	Limit on degenerate simplex moves	{-1, 0, 1, 5, 10, 20}	-1
	Presolving		
`Presolve`	Presolve aggressiveness	{-1, 0, 1, 2}	-1
`PrePasses`	Maximum number of presolve passes	{-1, 0, 1, 2, 5, 10, 20}	-1
`Aggregate`	Presolve aggregation level	{0, 1, 2}	1
`PreSparsify`	Presolve sparsification level	{-1, 0, 1, 2}	-1
`Symmetry`	Symmetry-detection level	{-1, 0, 1, 2}	-1
	LP and node relaxation		
`Method`	Root-relaxation solution method	{-1, 0, 1, 2}	-1
`NodeMethod`	Node-relaxation solution method	{-1, 0, 1, 2}	-1
`ScaleFlag`	Model-scaling strategy	{-1, 0, 1, 2, 3}	-1

Table 7: GNN feature summary used for our bipartite graph instance representation.

Node/Edge Type	**Feature Name**	**Description**	**Data Type**
	Variable Node Features		
Variable	`type`	One-hot encoding of variable type (continuous, binary, integer)	Binary vector
Variable	`coef`	Objective function coefficient (c_j)	Real
Variable	`has_bound`	Indicators for whether finite lower and upper bounds exist	Binary vector
Variable	`bound_val`	Values of the lower and upper bounds (if finite)	Real vector
	Constraint Node Features		
Constraint	`rhs`	Right-hand side value of the constraint (b_i)	Real
Constraint	`obj_cos`	Cosine similarity with the objective function vector	Real
	Edge Features		
Edge	`matrix_coef`	Non-zero coefficient in the constraint matrix (A_{ij})	Real

Table 8: Hyperparameters for the GNN selector architecture and training.

Category	Hyperparameter	Value
Training Optimization		
Training	Max Epochs	1000
Training	Batch Size	4 instances per batch
Training	Optimizer	Adam
Training	Learning Rate	1×10^{-4}
Training	LR Schedule	StepLR (step size = 50 epochs, $\gamma = 0.9$)
Training	Loss Function	Smooth L1 Loss ($\beta = 1$)
HGAT Architecture		
Model	Attention Variant	GATv2 (Heterogeneous)
Model	Hidden Dimension (d_h)	64
Model	Attention Heads	4 (head dimension = 16)
Model	Message-Passing Layers (L)	3
Model	Edge Feature Dimension	1 (scaled matrix coefficient)
Model	Normalization	GraphNorm (applied after each layer)
Model	Global Pooling	Sum over variables + Sum over constraints (concatenated)
Model	MLP Hidden Layers	$[128, 32]$
Model	Output Dimension	$K = \|\mathcal{P}\|$ (portfolio size)

to update the portfolio. Table 9 groups these evaluations into a single discovery-evaluation stage. Available evaluation records are reused through caching.

Label generation and selector training. The final portfolio's solving times on the training and validation sets provide supervision for the selector. GNN training reads cached labels without additional solver runs, and validation selects the model checkpoint. Training for 1,000 epochs on 100 `load` instances takes 39–47 minutes per model, with a mean of 43.4 minutes.

Table 9: Computational cost on `load`. Configuration evaluations and label generation use 20 parallel single-thread CPU workers. SMAC search and GNN training are reported separately in the text.

Stage	Elapsed time
Discovery evaluations: initialization and five updates	SCIP: 3.76 h; Gurobi: 3.8 min
Training-label generation	SCIP: 3.31 h; Gurobi: 2.0 min
Validation-label generation	SCIP: 3.61 h; Gurobi: 1.6 min
Online configuration selection	Measured mean on `load`: 0.24 s per instance

LLM usage. A complete five-round discovery procedure uses one initialization call and five refinement calls. Based on saved prompt and response lengths, we estimate the cost of one initialization plus five refinement calls on `load` at approximately $0.16 for SCIP and $0.15 for Gurobi.

Online configuration selection. Deployment requires neither additional LLM calls nor configuration search or GNN training. For each new instance, LEAP constructs its graph representation, performs one GNN forward pass, and runs the solver with the selected configuration. In the measured `load` setting, configuration selection takes approximately 0.24 seconds per instance on average, including approximately 0.20 seconds for instance reading and graph construction and 0.04 seconds for the forward pass. This excludes the subsequent solver run.

A.4 Experiment Details of Baselines

Configuration-search baselines receive the same ten-hour SMAC budget per benchmark. LLM-Cold-Start performs no SMAC search. Auxiliary evaluation and learning costs are accounted for separately, as described in Appendix A.3.

A.4.1 Baselines for Overall Comparison

Default solvers (Default). SCIP 9.0.2 or Gurobi 12.0.3 with its out-of-the-box parameter configuration, serving as the reference for relative improvements.

DGCAC. Deep Graph Clustering based Algorithm Configuration (DGCAC) Song et al. (2023) represents a state-of-the-art instance-specific AC approach. DGCAC models MIP instances as bipartite graphs and employs an auto-encoder to learn dense feature embeddings in an unsupervised manner. These embeddings are then clustered using the k-means algorithm to form homogeneous instance groups, followed by SMAC tuning for each cluster. We adapt DGCAC to each evaluated solver and use the corresponding solver's joint parameter space. We use five clusters and allocate two hours of SMAC search to each cluster, giving a total search budget of ten hours.

LLM-Cold-Start. Following the validation-based approach of Lawless et al. (2025), the LLM proposes ten configurations using problem descriptions and solver-parameter information. We choose the configuration with the lowest mean validation cost and apply that same configuration to every test instance. This baseline performs neither SMAC search nor instance-wise GNN selection.

Global-SMAC. SMAC searches for one configuration that minimizes mean cost over $\mathcal{D}_{\text{search}}$, using the full ten-hour budget. The resulting single configuration is fixed for test evaluation.

LEAP w/o Init & Refine. This variant corresponds to Hydra-style portfolio discovery Xu et al. (2011). The standard iterative discovery baseline starts from the corresponding solver's Default configuration and performs five sequential SMAC iterations of two hours each, without LLM initialization or space refinement.

LEAP w/o Refine. An ablation variant of our method. It utilizes the LLM to generate the initial portfolio of 10 configurations (identical to our full method) but bypasses the iterative configuration space refinement.

LEAP w/o Init. This variant starts from Default and uses LLM-driven space refinement between numerical searches, without LLM initialization. Together with LEAP w/o Init & Refine, LEAP w/o Refine, and full LEAP, it forms the four-way component ablation.

Portfolio evaluation (PI). PI reports the mean of the per-instance minimum measured solving costs within a portfolio. It assesses portfolio quality independently of learned selection and provides a lower bound on solving time for that portfolio on the evaluated instances.

A.5 Additional Evaluation Protocols

Generalization to larger instances (RQ3). We transfer from the six-week `mix_2` source domain to the 13- and 21-week target groups described in Appendix B.1. We evaluate LEAP and LEAP w/o Init & Refine using Gurobi 12.0.3. For each method and random seed, we directly reuse the portfolio discovered on the source domain under the standard five-iteration, ten-hour SMAC search budget and its corresponding GNN checkpoint selected on the source validation set. Both the portfolio and the GNN remain fixed throughout target evaluation. No additional LLM calls, configuration search, GNN training, or model selection are performed on the target instances.

The 13- and 21-week groups use relative optimality gaps of 1% and 5%, respectively, with a 500-second solving limit for every configuration, including Default. Runs that reach the limit are assigned a cost of 500 seconds. We separately report Default, PI, and GNN solving times for each group. PI uses the best measured configuration cost for each instance within the fixed portfolio, assessing portfolio quality independently of selection. GNN performance uses the measured solving cost of the configuration selected by the source-trained model. For each of three random seeds, we average solving times over the nine instances in each group; Table 3 reports the mean and standard deviation across these three seed-level averages.

Effect of discovery iterations (RQ4). We compare five, seven, and nine discovery iterations on SCIP `load` and `rhs_3`. Each iteration uses two hours of SMAC search, giving total search budgets of ten, fourteen, and eighteen hours. For each resulting portfolio, a separate GNN is trained using the settings in Appendix A.2, with its checkpoint selected on the validation set. Each setting uses three random seeds, and we report mean test solving time and improvement over Default under the standard evaluation protocol.

B BENCHMARKS

To evaluate our framework, we follow a consistent data utilization protocol across all datasets:

- **Training Set:** The training set also serves as the discovery set, i.e., $\mathcal{D}_{\text{search}} = \mathcal{D}_{\text{train}}$. It is used for configuration search, portfolio updates, and refiner feedback, and subsequently for GNN training from precomputed configuration costs.
- **Validation Set:** This set selects the GNN checkpoint and, for LLM-Cold-Start, the single configuration to deploy.
- **Test Set:** This set is used solely for the final evaluation to report the performance of our framework on unseen instances.

To provide a comprehensive assessment across different difficulty levels and structural characteristics, we evaluate our framework on two categories of benchmarks: Re-optimization Bolusani et al. (2024) and BenLOC Li et al. (2025). Specifically, within the Re-optimization dataset, we further categorize the datasets into *Easy* and *Hard* groups based on their problem scale and the number of evolving components.

Re-optimization datasets. Table 10 summarizes the re-optimization benchmarks used in our experiments, including the size, domain/source, and which model components vary within each sequence. Here, #Vars denotes the total number of variables (integer variables in parentheses), and #Cons denotes the number of constraints. An *X* indicates that the corresponding component changes across instances in the sequence. To analyze performance across different difficulty levels, we categorize the datasets into easy and hard groups. Easy datasets involve only a single varying component (e.g., objective coefficients), and have relatively small problem sizes, with no more than $1{,}457$ integer variables. The hard group includes datasets with larger problem sizes, changes to the constraint matrix, or simultaneous changes to multiple input components.

Table 10: Re-optimization benchmark datasets: size, source domain, and varying components. #Vars: total number of variables (integer variables in parentheses). LO: variable lower bounds; UP: variable upper bounds; OBJ: objective coefficients; LHS: left-hand sides; RHS: right-hand sides; MAT: constraint matrix coefficients. *X* indicates that the corresponding component varies within the instance sequence.

Dataset	#Vars	#Constrs	Domain/Source	LO	UP	OBJ	LHS	RHS	MAT
			Easy datasets						
bnd_2	1758 (1457)	351	MIPLIB'17	X	X	–	–	–	–
bnd_3	1758 (1457)	351	MIPLIB'17	X	X	–	–	–	–
rhs_2	1000 (500)	1250	Synthetic MILP	–	–	–	–	X	–
rhs_4	1000 (500)	1250	Synthetic MILP	–	–	–	X	X	–
obj_1	360 (360)	55	Stochastic multi 0–1 knapsack	–	–	X	–	–	–
			Hard datasets						
rhs_3	63009 (63009)	507	MIPLIB'17	–	–	–	X	X	–
obj_2	747(355)	26159	MIPLIB'17	–	–	X	–	–	–
mix_1	90895(60146)	33438	Hydroelectric valley industrial use	–	–	X	–	X	–
mix_2	27712(400)	16288	Multilevel supply chain	X	X	–	–	X	X
mix_3	7973 (5186)	3558	Hydro unit commitment	X	X	X	X	X	X

After the table, we provide detailed descriptions of each dataset, including how instances were generated and which variations are introduced:

bnd_2: Based on csched007 (MIPLIB 2017). Instances are generated via random fixings of 15%–25% of the binary variables, selected uniformly with respect to the original instance.

bnd_3: Also based on csched007. Instances are generated via random fixings of 5%–20% of the binary variables (uniform selection).

rhs_2: Based on a synthetic MILP dataset. Instances are generated by taking a convex combination of two different RHS vectors.

rhs_4: Also based on the synthetic MILP dataset as in rhs_2, but using a different pair of RHS vectors for the convex combination.

obj_1: Based on a stochastic multiple binary knapsack dataset. We consider one scenario at a time, yielding a series in which roughly one third of the objective vector (corresponding to y-variables) varies across instances.

rhs_3: Based on glass4 (MIPLIB). Instances are generated by perturbing nonnegative LHS and RHS components via a discrete uniform distribution by up to $\pm 70\%$ of their values.

obj_2: Based on the instance ci-s4 from the MIPLIB 2017 benchmark library. The instances are generated via random perturbations and random rotations of the objective vector.

rhs_obj(mix_1): Based on a hydroelectric valley (Ain River) industrial use case in France. Varying inputs include electricity prices (in the objective function) and the varying flows of the different affluents (in the RHS vector) of the river Ain.

mat_rhs_bnd(mix_2): Based on the MILP formulation of the multilevel supply chain of a fictitious cell phone company.

mat_rhs_bnd_obj (mix_3): Based on the hydro unit commitment (HUC) problem, corresponding to mat_rhs_bnd_obj_series_1 in the competition benchmark Bolusani et al. (2024). Every input component (objective, bounds, sides, and matrix) may vary.

BenLOC datasets. To comprehensively evaluate our proposed method, we benchmark our approach using the standardized environments provided by BenLOC Li et al. (2025), a comprehensive benchmark designed for learning to configure MIP optimizers. While BenLOC includes simpler tasks like Set Cover, we focus our analysis on load_balancing and nn_verification. These two datasets are chosen for their representativeness of real-world industrial challenges.

load_balancing: Originally introduced in the NeurIPS 2021 Machine Learning for Combinatorial Optimization (ML4CO) competition Gasse et al. (2022), this dataset deals with apportioning workloads robustly, ensuring valid assignments even in the event of worker failures. Each instance is mathematically modeled as a MILP utilizing a bin-packing formulation. We utilize a subset of 300 instances from the BenLOC dataset. This subset comprises 100 instances from each of the predefined training, validation, and testing splits to preserve the original data distribution.

nn_verification: Introduced by Cheng et al. (2017), this dataset focuses on the formal verification of neural networks. The core objective is to verify whether an artificial neural network is robust against input perturbations, which is precisely encoded as a MIP problem. We utilize a subset of 300 instances from the BenLOC dataset. This subset comprises 100 instances from each of the predefined training, validation, and testing splits to preserve the original data distribution.

B.1 Cross-Scale Generalization Instances

Source and target instances. We use the six-week mix_2 instances from the SAP mobile-phone supply-chain MILP benchmark as the source domain, corresponding to its vary_matrix_rhs_bounds dataset. We select larger instances from its official public_mps collection as the target domain. The 13- and 21-week groups contain six and ten discrete decision periods, respectively. Within each group, customer counts $\{35, 50, 65\}$ are combined with batch factors $\{1, 2, 3\}$, giving nine instances per group and eighteen target instances in total. This setting tests transfer under joint changes in planning horizon, discrete decision range, customer count, and batch factor.

The transfer and evaluation protocol is described in Appendix A.5.

C LEAP PORTFOLIO DISCOVERY ALGORITHM

Algorithm 1 instantiates the four steps in Section 3 with the Initializer and Refiner described in Section 4. For all portfolio evaluations, $c(\mathcal{P}, x) = \min_{s \in \mathcal{P}} c(s, x)$; previously measured configuration costs are reused.

Algorithm 1 LEAP Portfolio Discovery Algorithm

Input: Fixed base configuration space $\mathcal{S}$, problem context and parameter semantics, $\mathcal{D}_{\text{search}}$, cost c, configurator SMAC, LLM, initial candidate count N_{init}, rounds R, per-round search budget τ
Output: Configuration portfolio $\mathcal{P}^{(R)}$
Step 1: Portfolio initialization
$\mathcal{P}^{(0)} \leftarrow \{s_{\text{default}}\}$; evaluate Default on $\mathcal{D}_{\text{search}}$ and initialize $\mathcal{H}^{(0)}$
$(s_1^{(0)}, \ldots, s_{N_{\text{init}}}^{(0)}) \leftarrow \text{Init}_{\text{LLM}}(\text{problem context, parameter semantics}, \mathcal{S})$
for each complete and admissible initial candidate s in the returned order **do**
    Evaluate s on $\mathcal{D}_{\text{search}}$ and record costs in $\mathcal{H}^{(0)}$
    if $\exists x \in \mathcal{D}_{\text{search}} : \; c(s, x) < c(\mathcal{P}^{(0)}, x)$ **then**
        $\mathcal{P}^{(0)} \leftarrow \mathcal{P}^{(0)} \cup \{s\}$
    end if
end for
$\mathcal{S}^{(0)} \leftarrow \mathcal{S}$
for $r = 1$ **to** R **do**
    Step 2: Search-space refinement
    Query the Refiner with problem context, parameter semantics, $\mathcal{S}$, $\mathcal{S}^{(r-1)}$, $\mathcal{P}^{(r-1)}$, and $\mathcal{H}^{(r-1)}$
    Validate and apply its actions to obtain $\mathcal{S}^{(r)}$; invalid outputs leave $\mathcal{S}^{(r-1)}$ unchanged
    Step 3: Complementary configuration search
    $c_{\text{eval}}(s, x \mid \mathcal{P}^{(r-1)}) \leftarrow \min\{c(s, x), c(\mathcal{P}^{(r-1)}, x)\}$
    Run AC within $\mathcal{S}^{(r)}$ for budget τ to minimize $\sum_{x \in \mathcal{D}_{\text{search}}} c_{\text{eval}}(s, x \mid \mathcal{P}^{(r-1)})$; obtain candidate $s^{(r)}$ and retain search feedback
    Step 4: Portfolio and history update
    Evaluate $s^{(r)}$ on $\mathcal{D}_{\text{search}}$; $\mathcal{P}^{(r)} \leftarrow \mathcal{P}^{(r-1)}$
    if $\exists x \in \mathcal{D}_{\text{search}} : \; c(s^{(r)}, x) < c(\mathcal{P}^{(r-1)}, x)$ **then**
        $\mathcal{P}^{(r)} \leftarrow \mathcal{P}^{(r-1)} \cup \{s^{(r)}\}$
    end if
    Extend $\mathcal{H}^{(r-1)}$ with refinement actions, SMAC search feedback, and candidate evaluations to obtain $\mathcal{H}^{(r)}$, regardless of admission
end for
Sequentially remove redundant configurations from $\mathcal{P}^{(R)}$ while keeping it nonempty and preserving $c(\mathcal{P}^{(R)}, x)$ for every $x \in \mathcal{D}_{\text{search}}$
return $\mathcal{P}^{(R)}$

Refinement allows up to four actions per round: `focus` only in round 1, and `focus` or `expand` thereafter. A `focus` action retains a nonempty strict subset of the current values; an `expand` action retains all current values and restores choices from the fixed original space. Unmentioned parameters remain unchanged. Validation enforces valid parameter names, admissible values, and these set relationships, so $\varnothing \neq \mathcal{S}^{(r)} \subseteq \mathcal{S}$ throughout. The search budget τ covers configurator search; full discovery-set evaluations outside that search are accounted for separately in Appendix A.3.

D LLM PROMPT EXAMPLE

In this section, we provide representative prompt–response excerpts from our LLM-guided offline configuration discovery procedure. We use the SCIP `load_balancing` experiment as a running example because its dense assignment structure, strong symmetry, and robustness constraints provide a clear setting for illustrating solver configuration decisions. The following subsections present the two LLM-assisted stages: (i) the Initializer, which generates a diverse starting portfolio from the problem description and fixed master configuration space, and (ii) the Refiner, which uses measured portfolio and SMAC feedback to adapt the categorical working subspace searched in each subsequent round. Both stages are performed offline; the LLM is not invoked during online configuration selection or MILP solving.

D.1 LLM PROMPT EXAMPLE: CONFIGURATION PORTFOLIO INITIALIZATION

Initial Portfolio Discovery Prompt. The following example is taken from the SCIP experiment on the `load_balancing` family. The initializer receives the MILP formulation, the structural characteristics of the instance family, and the complete fixed master configuration space. It is asked to propose a diverse initial portfolio; it does not observe any solver measurements at this stage.

Prompt

(i) Task description.

- **System role:** You are an expert in MIP solvers and algorithm configuration.
- **Objective:** Analyze the supplied MILP family and generate exactly 10 diverse and complementary SCIP configurations for initializing a configuration portfolio.
- **Diversity requirement:** Cover distinct solver strategies, such as cut-oriented, heuristic-oriented, branching-oriented, presolve-oriented, and balanced configurations, while relating each strategy to the problem structure.
- **Validity requirement:** Use only parameters and categorical values from the supplied master space. Every configuration must contain every listed parameter exactly once.
- **Output format:** Return one JSON array only, with no prose outside JSON. Each item must contain `"id"`, `"strategy"`, `"rationale"`, and a complete `"config"`. If exactly 10 valid configurations cannot be produced, return `[]`.

(ii) Problem description.

- **Problem:** Robust load balancing with binary assignment decisions.
- **Dataset metadata:** The homogeneous family contains 2,286 instances. The combinatorial structure is fixed, whereas job workloads and worker capacities vary across instances. The discovery set contains 100 instances.
- **Structural characteristics:** Strong symmetry, a dense assignment matrix, bin-packing structure, and robustness constraints requiring feasibility after the loss of any single worker.
- **Objective:** Minimize the maximum normalized worker load, $\min_{x,L} L$.
- **Core constraints:**
  - job assignment: $\sum_{m \in \mathcal{M}} x_{j,m} = 1$, for every job j;
  - capacity: $\sum_{j \in \mathcal{J}} w_j x_{j,m} \leq C_m L$, for every worker m;
  - single-failure robustness constraints; and
  - domains: $x_{j,m} \in \{0, 1\}$ and $L \geq 0$.

(iii) Fixed master configuration space.

- The full prompt enumerates all 25 SCIP parameters, their descriptions, admissible categorical values, and defaults (Table 5).
- Representative entries are shown below:

- `separating/aggregation/freq`: allowed `[-1, 0, 1, 5, 10, 20]`, default `10`;
- `separating/clique/freq`: allowed `[-1, 0, 1, 5, 10, 20]`, default `0`;
- `branching/scorefunc`: allowed `["p", "s", "q"]`, default `"p"`;
- `branching/scorefac`: allowed `[0.0, 0.167, 0.5, 0.75, 1.0]`, default `0.167`;
- `heuristics/rins/freq`: allowed `[-1, 0, 1, 5, 10, 15, 20, 25]`, default `15`; and
- `presolving/maxrounds`: allowed `[-1, 0, 1, 5, 10, 20]`, default `-1`.

Required output schema.

- One JSON array containing exactly 10 items.
- Keys per item: `"id"`, `"strategy"`, `"rationale"`, and `"config"`.
- Each `"config"` contains all 25 parameters and only admissible master-space values.

LLM Initializer Output. The following is one item from the actual 10-configuration response. The branching-oriented strategy is motivated by the dense, symmetric assignment structure. All values belong to the fixed SCIP master space.

Response


```
[
  {
    "id": "branching_focused_3",
    "strategy": "branching-focused",
    "rationale": "Optimize branching decisions for the dense
    ↪  assignment matrix.",
    "config": {
      "separating/aggregation/freq": 10,
      "separating/clique/freq": 10,
      "separating/mcf/freq": 10,
      "separating/impliedbounds/freq": 10,
      "separating/cmir/freq": 10,
      "separating/strongcg/freq": 10,
      "separating/zerohalf/freq": 10,
      "separating/gomory/freq": 10,
      "separating/cgmip/freq": -1,
      "separating/oddcycle/freq": -1,
      "separating/rapidlearning/freq": 5,
      "branching/scorefunc": "q",
      "branching/scorefac": 1.0,
      "branching/preferbinary": true,
      "branching/lpgainnormalize": "d",
      "branching/delaypscostupdate": false,
      "heuristics/trivial/freq": 5,
      "heuristics/rins/freq": 10,
      "heuristics/proximity/freq": 5,
      "heuristics/pscostdiving/freq": 5,
      "heuristics/randrounding/freq": 5,
      "heuristics/alns/freq": 10,
      "presolving/maxrounds": 5,
      "presolving/maxrestarts": 5,
      "presolving/abortfac": 0.1
    }
  },
  ... nine additional configurations omitted for space ...
```

```
]
```

D.2 LLM Prompt Example: Configuration-Space Refinement

Adaptive Working-Subspace Refinement Prompt. Before each portfolio-construction round, the refiner selects a categorical working subspace; it does not directly select a solver configuration. Standard SMAC subsequently searches within the selected subspace. The example below is the recorded SCIP `load_balancing` call for seed 2, round 3. For readability, we show selected context and summary fields rather than the complete Refiner input.

Prompt

(i) Task description.

- **Objective:** Select the categorical working subspace for the next LEAP portfolio-construction round.
- **Fixed master-space rule:** Never create parameters or values. The same master space is shared by all methods.
- **Action budget:** Return between zero and four actions. In round 1, only `focus` is permitted; from round 2 onward, both `focus` and `expand` are permitted.
- **Focus:** The proposed values must be a non-empty strict subset of the parameter's current values.
- **Expand:** The proposed values must be a strict superset of the current values and a subset of the master values. Expansion must cite plateau, conflicting, or boundary-best evidence.
- **Evidence rule:** Read `evidence_quality` before reasoning and treat per-value statistics as associative rather than causal evidence. If comparisons are uninformative, an action may rely only on an explicitly identified domain hypothesis.
- Unmentioned parameters remain unchanged. Never modify the cutoff, gap, thread count, tolerances, resource limits, stopping criteria, or logging.

(ii) Recorded experiment context.

```
{
  "solver": "SCIP",
  "hydra_round_to_run": 3,
  "cutoff_seconds": 500.0,
  "gap": 0.05,
  "n_discovery_instances": 100,
  "objective": "mean_i min(current_portfolio_runtime_i,
  ↪  candidate_runtime_i); lower is better"
}
```

(iii) Evidence summary supplied to the refiner.

- Twelve measured non-default configurations, eight of which improved the portfolio, producing nine distinct portfolio means.
- Current portfolio mean: 123.681313 seconds; six of the 100 discovery instances remain at the 500-second cutoff.
- The preceding SMAC round made 245 target calls and evaluated 82 unique configurations.
- `comparative_outcomes_uninformative = false`; measured comparisons are available, although per-value associations are not causal.

(iv) Relevant current and master values.

```
{
  "separating/aggregation/freq": {
    "current_values": [0, 1, 5],
    "master_values": [-1, 0, 1, 5, 10, 20]
```

```
  },
  "branching/scorefunc": {
    "current_values": ["p", "s"],
    "master_values": ["p", "s", "q"]
  }
}
```

Required output schema.

```
{"actions":[{"parameter":"exact name",
             "operation":"focus|expand",
             "values":[],
             "reason":"short evidence-based reason",
             "evidence":["context.path"]}],
 "round_rationale":"short"}
```

LLM Refiner Output (Seed 2, Round 3). The recorded response performs two reversible expansions. It introduces no new categorical value outside the fixed base configuration space. To avoid overlong verbatim lines, the free-text reasons and evidence paths below are shortened for display.

Response

```
{
  "actions": [
    {
      "parameter": "separating/aggregation/freq",
      "operation": "expand",
      "values": [0, 1, 5, 10],
      "reason": "Plateau evidence supports testing a higher
      ↪  frequency.",
      "evidence": ["recent_smac_evidence...aggregation/freq"]
    },
    {
      "parameter": "branching/scorefunc",
      "operation": "expand",
      "values": ["p", "s", "q"],
      "reason": "Boundary-best and conflicting evidence support
      ↪  restoring 'q'.",
      "evidence": ["recent_smac_evidence...branching/scorefunc"]
    }
  ],
  "round_rationale": "Expand using plateau and boundary evidence."
}
```

E Additional Experiment Results

E.1 Portfolio Quality on the Test Set

We further evaluate the final portfolios using PI on the test set. Unlike RQ2, which examines the discovery process on $\mathcal{D}_{\text{search}}$, this evaluation fixes each portfolio after discovery and takes its lowest measured solving time for each test instance. It therefore excludes selector errors and directly assesses portfolio quality on unseen instances. Table 11 reports mean PI times and sample standard deviations over three random seeds.

Portfolio gains from LLM guidance extend to unseen instances. Compared with the variant without either LLM component, full LEAP achieves lower PI times on 11 of 12 SCIP benchmarks and all 12 Gurobi benchmarks. On `load`, full LEAP outperforms both single-component variants for both solvers. Although the benefits of each component vary across benchmarks, these test-set results support the contributions of knowledge-guided initialization and feedback-driven space refinement to portfolio quality, independently of whether a downstream selector fully realizes those gains.

Table 11: Test-set PI time (s) of the final configuration portfolios. Portfolio results are means over three seeds, with std. dev. in parentheses. Default reports the default solver's mean test time, as in the main tables. Lower is better; bold marks the lowest unrounded mean among the four methods. The w/o columns denote LEAP ablations; Both means Init and Refine.

Dataset	Default	w/o Both	w/o Refine	w/o Init	LEAP
			SCIP		
bnd_2	248.7	193.4 (26.9)	**173.6 (18.1)**	218.6 (10.9)	179.0 (11.7)
bnd_3	420.2	383.0 (24.4)	**370.4 (12.9)**	385.9 (16.0)	384.4 (4.4)
rhs_2	92.1	58.7 (2.2)	52.1 (2.5)	58.0 (0.6)	**50.7 (2.3)**
rhs_4	89.5	59.4 (2.7)	60.3 (2.3)	58.9 (2.4)	**55.8 (3.7)**
obj_1	465.8	377.3 (10.1)	399.5 (11.5)	**371.3 (41.1)**	374.9 (28.2)
rhs_3	325.4	194.0 (1.8)	158.4 (3.5)	198.4 (9.0)	**147.8 (6.9)**
obj_2	392.3	356.5 (10.3)	322.9 (5.6)	353.2 (11.9)	**316.1 (18.5)**
mix_1	191.0	67.2 (19.8)	41.9 (3.3)	63.0 (13.7)	**39.7 (1.9)**
mix_2	285.7	212.9 (18.4)	210.9 (2.7)	202.5 (25.5)	**177.8 (13.0)**
mix_3	98.0	66.2 (6.2)	**57.5 (4.2)**	64.7 (7.4)	58.3 (5.2)
nn	77.4	39.7 (3.3)	32.4 (3.7)	36.0 (9.0)	**30.5 (3.1)**
load	250.7	158.8 (19.0)	150.2 (17.3)	144.5 (0.9)	**130.2 (2.7)**
			Gurobi		
bnd_2	271.5	270.2 (10.3)	270.3 (13.8)	269.3 (6.6)	**267.0 (12.0)**
bnd_3	277.0	278.4 (24.2)	261.2 (24.1)	277.3 (25.6)	**258.5 (24.7)**
rhs_2	14.2	6.6 (0.7)	6.0 (0.4)	6.1 (0.4)	**5.9 (0.3)**
rhs_4	15.1	6.2 (0.6)	5.9 (0.3)	5.8 (0.5)	**5.2 (0.3)**
obj_1	109.4	107.0 (4.4)	**98.8 (1.7)**	105.8 (4.0)	100.2 (6.5)
rhs_3	23.7	7.2 (0.9)	**5.2 (0.6)**	6.9 (0.7)	5.7 (1.4)
obj_2	27.7	2.9 (3.9)	2.9 (3.7)	**1.2 (0.7)**	1.4 (0.7)
mix_1	12.6	10.7 (0.1)	10.3 (1.0)	10.7 (1.2)	**10.2 (0.5)**
mix_2	59.0	4.6 (0.9)	3.5 (0.4)	5.8 (1.0)	**3.5 (0.3)**
mix_3	9.5	4.3 (0.8)	4.0 (1.0)	4.3 (1.3)	**3.8 (0.3)**
nn	6.2	2.4 (0.1)	2.3 (0.3)	2.2 (0.6)	**2.1 (0.3)**
load	14.9	0.7 (0.0)	0.7 (0.0)	0.7 (0.0)	**0.6 (0.1)**

E.2 SOLVE RATES

To examine performance on challenging benchmarks, we focus on the seven SCIP datasets where Default solves fewer than 90% of test instances within the 500-second limit. Default solve rates range from 39.7% to 84.0%, indicating that a substantial fraction of instances remain unsolved under the prescribed optimality tolerances. Table 12 reports the percentage of test instances solved within 500 seconds, averaged over three random seeds, with sample standard deviations. Success requires a 5% relative optimality gap for `load` and optimality for the other benchmarks. LEAP achieves higher solve rates than Default on all seven reported SCIP benchmarks. For example, solve rates increase from 84.0% to 97.3% on `mix_1` and from 42.0% to 66.7% on `obj_1`, improvements of 13.3 and 24.7 percentage points, respectively. LEAP also achieves higher solve rates than Global-SMAC on six of these benchmarks. These results show that LEAP not only reduces average solving time but also enables more instances to be solved within the same time limit.

Table 12: Test-set solve rate (%) under the 500-second limit on SCIP. Values are means over three seeds, with std. dev. in parentheses; higher is better. Bold marks the highest mean solve rate in each row.

Dataset	Default	DGCAC	Global-SMAC	LEAP
		SCIP		
`bnd_3`	39.7 (7.5)	44.0 (0.0)	42.7 (10.5)	**45.3 (8.3)**
`obj_1`	42.0 (3.5)	62.7 (10.1)	62.7 (2.3)	**66.7 (13.1)**
`rhs_3`	64.0 (0.0)	62.7 (4.6)	58.7 (4.6)	**64.7 (6.1)**
`obj_2`	60.0 (3.6)	54.7 (6.1)	**69.3 (2.3)**	65.7 (6.2)
`mix_1`	84.0 (0.0)	92.0 (6.9)	92.0 (6.9)	**97.3 (2.3)**
`mix_2`	80.0 (0.0)	74.7 (2.3)	80.0 (0.0)	**82.0 (6.0)**
`load`	64.0 (0.0)	62.5 (3.4)	67.7 (6.4)	**68.3 (2.1)**

F LIMITATIONS

- **Configuration-space coverage.** LEAP is evaluated within predefined discrete configuration spaces. The Refiner adjusts the working subspace only within these fixed boundaries for fair comparison with baselines; it does not introduce new parameters or values. Extending discovery to broader parameter sets and finer-grained domains remains an important direction for future work.
- **Offline computational cost.** LLM guidance does not replace solver evaluations. More expensive instances and larger discovery sets can therefore increase the cost of constructing a portfolio. These costs are incurred offline; deployment requires neither further LLM calls nor configuration search. Appendix A.3 reports the discovery and deployment costs separately.
- **Scope of generalization.** Our cross-scale evaluation transfers portfolios and selectors within the `mix_2` problem family. These results support transfer to larger instances in this setting, but do not establish generalization to arbitrary problem families or distribution shifts. Broader transfer evaluations remain necessary.