LEGO: LLM-based Evaluation and Guided Optimization for Adaptive Algorithm Design

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

To efficiently solve optimization problems, a wide variety of algorithms have been developed, each designed to perform well under specific problem structures or domains. However, due to the no-free-lunch theorem, no single algorithm consistently outperforms others across all instances. This raises a fundamental question: **how can we automatically construct algorithms tailored to the given problem?** Inspired by LEGO-style modularity, we propose LEGO, a general framework for adaptive algorithm design via pipeline component selection. Using Mixed Integer Linear Programming (MILP) as a prototype, we construct solving pipelines by selecting and configuring components and hyperparameter within the Predict-and-Search paradigm. To ensure adaptability across varying data scales, LEGO can self-adaptively generate synthetic datasets of different sizes, enabling robust configuration even with limited data. It leverages large language models (LLMs) to evaluate and guidedly optimize candidate configurations, using a hybrid metric that combines classical performance indicators with LLM-informed assessments. High-quality pipelines are selected through hypervolume-based ranking and further refined via performance transfer on synthetic data to improve scalability. Experiments on four benchmark MILP tasks demonstrate that the proposed evaluation framework effectively identifies high-performing strategies and hyperparameter configurations, leading to algorithms that are both more efficient and more effective, highlighting LEGO as a generalized framework for component and hyperparameter selection in MILP solving frameworks, with potential for extension to broader algorithm design.

1 Introduction

Designing efficient algorithms for solving hard optimization problems is a core challenge across scientific and engineering domains. Over the years, a wide variety of algorithms have been proposed, each tailored to specific problem structures or domains [1, 2, 3]. However, the *no-free-lunch theorem* implies that no single algorithm can consistently outperform all others across the full spectrum of problem instances [4]. This raises a fundamental question: **how can we automatically construct algorithms that are tailored to a given problem?**

In this paper, we address this question by proposing **LEGO**, a general framework for adaptive algorithm design via modular pipeline construction and optimization. Inspired by the flexibility of LEGO-style block assembly, LEGO treats algorithm design as a combinatorial construction problem—selecting and configuring components and hyperparameters within a parameterized framework. While the core ideas are broadly applicable, we instantiate and validate LEGO in the domain of *Mixed Integer Linear Programming* (MILP), a widely adopted modeling paradigm for combinatorial optimization tasks such as routing [5], scheduling [6], and supply chain optimization [7].

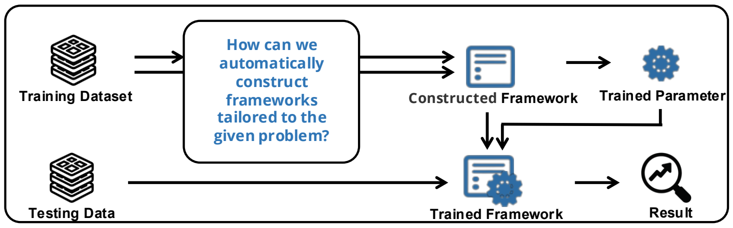

Figure 1: Core research question: Given a problem distribution, how can we construct a solving framework tailored to it? LEGO automatically assembles and tunes a solving pipeline based on training data, and transfers it to larger test instances.

To ground our investigation, we build upon the *Predict-and-Search* (P&S) paradigm [8, 9], which structures MILP solvers into modular stages such as graph embedding, neural prediction, solution repair, and heuristic search. Existing P&S-based frameworks often commit to fixed designs and fixed-size data configurations, limiting their adaptability. In contrast, LEGO decomposes the P&S architecture into interchangeable components and systematically explores the space of configurations, enabling automated construction of solving pipelines tailored to the task at hand.

A key challenge in this process is how to evaluate and select from a large pool of candidate solver configurations. To this end, LEGO introduces an **LLM-enhanced performance evaluation** mechanism, which combines classical indicators (e.g., objective gap, runtime) with LLM-informed metrics to assess solving behavior in a more holistic and task-aware way. These evaluations are then aggregated using a **hypervolume-based component selection** strategy to identify high-quality solver pipelines under multiple objectives. Furthermore, LEGO supports generalization across problem scales. Even when only small-scale training data are available, LEGO leverages a **scalable data generation** module to synthesize large-scale instances with similar structure, enabling robust **size-transferable parameter tuning**. This ensures that the selected pipelines can be effectively adapted to larger and more complex problems, even under limited training dataset. A demo of LEGO is available at `https://anonymous.4open.science/r/LEGO-B36D`, and the full codebase will be released after the review process.

Our contributions are summarized as follows:

- **Scale-aware adaptive framework construction.** LEGO introduces a flexible mechanism to generate synthetic instances of arbitrary scale with high structural similarity, enabling algorithm configuration even under data-scarce or size-mismatched scenarios.
- **A unified LLM-guided optimization framework.** We propose three key techniques to guide the construction of solving pipelines: (1) **LLM-enhanced Performance Evaluation**, which combines classical metrics with LLM-informed assessments; (2) **Hypervolume-guided Component Selection**, which enables robust multi-objective ranking; and (3) **Size-transferable Parameter Tuning**, which refines pipeline performance across scales.
- **Empirical validation on MILP benchmarks.** We validate LEGO on four widely used MILP benchmarks (MIS, MVC, SC, MKS). Results show that LEGO consistently discovers high-performing solver frameworks, outperforming both classical solvers (e.g., Gurobi, SCIP) and ML-based baselines (e.g., Light-MILPopt).

2 Related Work

2.1 Automatically Algorithm Design

Automatically Algorithm Design (AAD) seeks to construct or adapt algorithms to specific problem distributions, fundamentally motivated by the "no-free-lunch" theorem, which states that no single algorithm excels on all problems. Classical AAD paradigms include *Algorithm Configuration* (AC) [10, 11], which optimizes hyperparameters for a fixed algorithm (e.g., ParamILS, LEAPSAND-BOUNDS), and *Algorithm Selection* (AS) [12], which chooses the most suitable algorithm from a predefined portfolio based on instance features. These approaches are often formalized under the *Meta-Black-Box Optimization* (MetaBBO) framework [13], where meta-level learning drives algorithmic adaptation.

While effective in many settings, these methods typically treat algorithms as atomic units, limiting flexibility and reusability [14]. Recent trends move toward more granular, component-based AAD [15], where algorithms are assembled from interchangeable functional units. This modular design enables finer control and potentially richer adaptation. However, challenges remain: most existing systems struggle with generalization across problem scales[16], as configurations optimized on one size often degrade on others. Moreover, managing inter-component dependencies and avoiding performance bottlenecks in dynamic or large-scale settings remains a significant open problem.

2.2 Mixed Integer Linear Programs

Mixed Integer Linear Programs (MILPs) are a fundamental class of combinatorial optimization problems, defined by a linear objective function with linear constraints, where some variables are restricted to take integer values [17]. The general form of an MILP is given by:

$$\min_{x} \ c^T x \quad \text{s.t.} \quad Ax \le b, \ l \le x \le u, \ x_i \in \mathbb{Z} \text{ for } i \in \mathbb{I}, \tag{1}$$

where $x \in \mathbb{R}^n$ denotes the decision variables, $c \in \mathbb{R}^n$ the objective coefficients, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ define the linear constraints, and $l, u \in \mathbb{R}^n$ represent variable bounds. The index set $\mathbb{I} \subseteq \{1, \dots, n\}$ indicates the subset of variables that must take integer values.

Solving MILPs is NP-hard in general [18], and exact methods such as branch-and-bound, branch-and-cut, and cutting plane techniques remain the backbone of modern solvers [17]. Despite the success of commercial tools like Gurobi and open-source solvers like SCIP, large-scale or real-time MILPs often remain computationally intractable. Recent efforts have explored hybrid approaches, combining classical methods with learning-based components to improve scalability and adaptability [19], giving rise to modular frameworks such as the *Predict-and-Search* paradigm.

2.3 Predict-and-Search

The Predict-and-Search (P&S) paradigm [8, 9] offers a flexible framework that integrates learning-based modules into traditional optimization solvers by structuring them into distinct stages—typically involving a prediction phase followed by a search phase. This modular decomposition facilitates the injection of data-driven components to tailor solver behavior. However, both the prediction and search stages admit a wide range of possible designs—e.g., different neural architectures, scoring heuristics, or branching rules—leading to a large combinatorial space of solver configurations. Moreover, the performance of a given configuration often varies significantly across problem scales or distributions. These factors highlight the need for automatic algorithm design methods that can adaptively select and compose solver components, while ensuring robustness and generalization across diverse instances.

3 Method

We propose **LEGO**, a general framework for adaptive algorithm design within a fixed problem domain. LEGO automatically constructs high-performance algorithms tailored to a specific class of optimization problems by assembling solving pipelines from modular components and optimizing them via LLM-guided evaluation and search. To support adaptation across instance scales, LEGO integrates a synthetic instance generator that enables effective pipeline tuning even with limited training data.

As shown in Figure 2, LEGO consists of two main modules. The upper module—**LLM-based Evaluation and Guided Optimization**—serves as the general optimization engine, combining hybrid metric evaluation with multi-objective search to guide pipeline construction. To demonstrate the framework's effectiveness, we instantiate LEGO for *Mixed Integer Linear Programming* (MILP), building a **Component Library** based on the Predict-and-Search (P&S) paradigm. This library includes interchangeable components such as graph embeddings, neural predictors, repair heuristics, and search strategies. LEGO composes and tunes these components to construct scalable and adaptive MILP solvers.

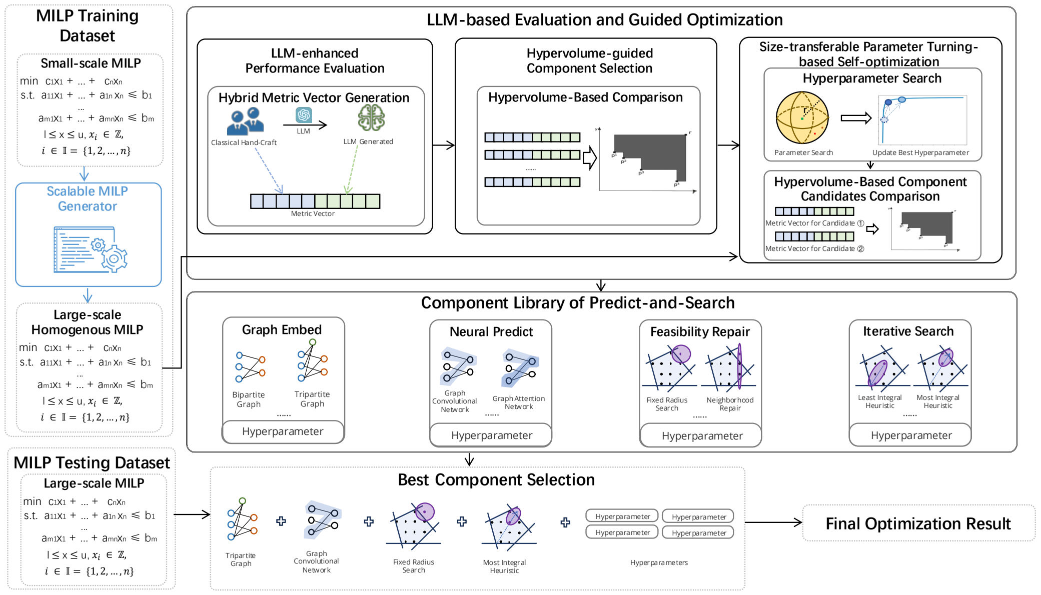
MILP Training Dataset
Small-scale MILP
Scalable MILP Generator
Large-scale Homogenous MILP
LLM-based Evaluation and Guided Optimization
LLM-enhanced Performance Evaluation
Hybrid Metric Vector Generation
Classical Hand-Craft
LLM
LLM Generated
Metric Vector
Hypervolume-guided Component Selection
Hypervolume-Based Comparison
Size-transferable Parameter Turning-based Self-optimization
Hyperparameter Search
Parameter Search
Update Best Hyperparameter
Hypervolume-Based Component Candidates Comparison
Metric Vector for Candidate ①
Metric Vector for Candidate ②
Component Library of Predict-and-Search
Graph Embed
Bipartite Graph
Tripartite Graph
Neural Predict
Graph Convolutional Network
Graph Attention Network
Feasibility Repair
Fixed Radius Search
Neighborhood Repair
Iterative Search
Least Integral Heuristic
Most Integral Heuristic
Hyperparameter
MILP Testing Dataset
Large-scale MILP
Best Component Selection
Hyperparameters
Final Optimization Result

Figure 2: Overview of the proposed **LEGO** framework. Solving pipelines are optimized via **LLM-based Evaluation and Guided Optimization**, and assembled from the modular **Component Library** following the Predict-and-Search paradigm. Optional blue modules (e.g., the Scalable MILP generator) support adaptation to large-scale instances when training data is limited.

3.1 LLM-based Evaluation and Self-Optimization

To evaluate and optimize component combinations within LEGO, we design a unified framework that integrates multi-dimensional performance metrics, including both classical solver indicators and task-specific criteria automatically generated by large language models (LLMs). These *LLM-enhanced performance evaluations* enable context-aware, goal-aligned assessment of solver behavior. To compare diverse configurations, we adopt a *hypervolume-based selection* strategy that jointly considers multiple objectives. To reduce evaluation cost, candidate pipelines are first assessed on small-scale instances, then adapted to larger problems through *size-transferable parameter tuning*, enabling efficient self-optimization with generalization across instance scales.

3.1.1 LLM-enhanced Performance Evaluation

To comprehensively evaluate pipelines constructed from the LEGO component library, we adopt a hybrid metric framework that combines classical human-designed indicators with task-aware criteria generated by large language models (LLMs). Specifically, we define a set of seven evaluation metrics that capture both solution quality and search dynamics over time.

We first construct four classical metrics based on oracle-level performance. For each training instance, we run a strong solver for a long time to obtain a high-quality upper bound x^*, and evaluate pipelines by: (1) the initial solution gap to x^*, (2) the final solution gap to x^*, (3) the *efficiency rate*, defined as the fraction of instances with final gap $\leq$ 10%, and (4) the time to reach the first valid solution (gap $\leq$ 10%) and the first high-quality solution (gap $\leq$ 1%). To complement these, we employ GPT-4o [20] to generate candidate evaluation criteria from solver trajectories. After filtering and human validation, we retain three additional metrics: (5) the solution gap at 20% of the time budget, (6) the gap at 60% of the time budget, and (7) the time-integrated solution gap over the full horizon (i.e., area under the gap-time curve). Details on prompt design, filtering strategy, and implementation are provided in the appendix.

The resulting 7-dimensional evaluation vector offers a fine-grained, multi-perspective assessment of solver performance, combining expert insight with LLM-generated domain knowledge to guide downstream optimization and selection.

3.1.2 Hypervolume-guided Component Selection

To efficiently select promising solver pipelines from the large combinatorial space of LEGO component configurations, we first evaluate all candidates on small-scale training instances using the *LLM-enhanced Performance Evaluation* described above, resulting in a 7-dimensional performance vector for each.

Due to the large number of configurations, we apply non-dominated sorting to identify the Pareto frontier S, which contains all configurations that are not strictly outperformed across all evaluation dimensions. This step filters out clearly suboptimal pipelines and retains those that offer distinct trade-offs in performance. However, the frontier S may still contain many similar or marginally different candidates. To further rank configurations within S, we compute the *hypervolume contribution* of each configuration $d \in S$:

$$\Delta V(d) = V(S) - V(S \setminus \{d\}),$$

where $V(\cdot)$ denotes the hypervolume with respect to a fixed reference point. A larger $\Delta V(d)$ indicates that configuration d contributes uniquely to the performance diversity of the Pareto set.

We rank all candidates in S by their hypervolume contribution and select the top-K configurations as solver pipelines for further tuning and deployment. This approach ensures efficient and diverse component selection, while keeping evaluation cost low by operating on small-scale instances.

3.1.3 Size-transferable Parameter Tuning

Since component selection is performed on small-scale training instances, we introduce a *Size-transferable Parameter Tuning* stage to ensure the resulting solver pipelines generalize effectively to larger-scale problems. This step jointly finalizes both component choices and their associated hyperparameters for deployment.

If the training dataset includes instances of varying sizes—especially those comparable to the test-time scale—we directly perform parameter tuning on the larger training instances. In the more common case where only small-scale data is available, we leverage a controllable instance generator based on MILP-retrieval [21], which can synthesize structurally similar problems with adjustable scale, difficulty, and similarity. This allows LEGO to adaptively tune solvers under any data regime.

In this stage, we apply Bayesian Optimization to search optimal hyperparameters for each of the top-K candidate configurations. The tuned candidates are then evaluated using the *LLM-enhanced Performance Evaluation* described earlier, and ranked using *Hypervolume-Based Component Candidates Comparison*. The final output is the best-performing configuration and its hyperparameters, optimized for both effectiveness and scalability.

3.2 Component Library

Building on the LLM-based evaluation and self-optimization framework introduced above, we apply LEGO to the important application domain of mixed-integer linear programming (MILP). Concretely, we instantiate LEGO within the widely adopted Predict-and-Search (P&S) paradigm, a dominant approach in learning-based MILP solving. Under this paradigm, LEGO decomposes the solver pipeline into four functional stages: *Graph Embed*, *Neural Predict*, *Feasibility Repair*, and *Iterative Search*. Each stage defines a modular interface with multiple candidate implementations, forming the LEGO Component Library. By systematically selecting and composing components across these stages, LEGO can generate diverse, adaptive solving frameworks tailored to different problem distributions and instance scales. A complete description of all components and their pseudocode is provided in the appendix.

3.2.1 Graph Embed

Directly feeding the raw algebraic form of a MILP problem into a neural model may obscure key structural invariances, such as row and column permutations that preserve problem equivalence. To retain these desirable symmetries, graph-based representations are commonly adopted, as they are inherently invariant to permutations of node order and layout. The role of the Graph Embed module is to transform a given MILP instance into a lossless graph representation that preserves its combinatorial and constraint structure.

In LEGO, we support several widely used encodings to capture the structure of MILPs: the *bipartite graph* representation [22], which connects variables and constraints as two disjoint node types; the *tripartite graph* representation [23], which further separates objective coefficients as a third node type; and two enhanced variants designed for foldable MILP instances: *bipartite with random feature strategy* and *tripartite with random feature strategy*, which inject randomized node features [24] to improve representation diversity in structurally repetitive problems called "foldable" problems. Each encoding defines a distinct component in the Graph Embed stage of our Component Library.

3.2.2 Neural Predict

Given the graph representation of a MILP instance, the Neural Predict module applies graph neural networks (GNNs) to learn mappings from problem structure to solution space. During training, a GNN is trained to predict optimal or near-optimal solutions based on the graph-structured input of MILP instances. At inference time, the network generalizes to unseen problems and provides predictions that serve as initial candidates or guidance for downstream search. LEGO integrates two widely used GNN architectures in this stage: the *Graph Convolutional Network* (GCN) [25] and the *Graph Attention Network* (GAT) [26], both featuring semi-convolutional designs to capture local and contextual structure. Network depth (i.e., number of layers) is treated as a tunable hyperparameter to support flexible capacity control during optimization.

For problem settings where feasible regions are extremely small, fragmented, or hard to learn, neural networks may struggle to produce high-quality or even feasible predictions. To address this, LEGO also supports solver-based predictors using commercial solvers such as *Gurobi* [27] and *SCIP* [28]. These solvers can generate initial feasible solutions reliably, even when neural predictors fail or are uncertain. Although solver-based initializations may be suboptimal compared to learned predictions in many cases, they offer robustness in challenging problem domains. LEGO thus enables hybrid designs within the Neural Predict stage, where learning-based and solver-based components can be used individually or in combination, depending on the characteristics of the target problem.

3.2.3 Feasibility Repair

The predictions generated by the Neural Predict module are not guaranteed to satisfy all constraints of the original MILP problem. Directly using such infeasible solutions in downstream search often leads to inefficient or invalid trajectories. To address this, the Feasibility Repair module aims to transform potentially infeasible predictions into valid solutions that respect problem constraints.

LEGO integrates three complementary repair strategies. The first is the *adaptive radius search* [8], which defines a dynamic neighborhood around the predicted solution and invokes a solver to explore feasible candidates within this radius. This approach balances prediction guidance with combinatorial search flexibility. The second is the *adaptive threshold* method [29], which adjusts the prediction confidence threshold and delegates the unresolved portion of the solution to a lightweight solver. The third is the *neighborhood repair* strategy[30], which prunes the prediction set based on local constraint structures to reduce infeasibility. These strategies are implemented as interchangeable components in the repair stage. Detailed algorithmic descriptions can be found in the Appendix.

3.2.4 Iterative Search

Once a feasible solution is obtained, it can be further improved via iterative search, a widely adopted strategy in modern MILP solving frameworks. The core idea is to fix a subset of decision variables and iteratively refine a selected neighborhood using a solver. This process allows local exploration around promising solutions and can significantly enhance solution quality.

LEGO supports five distinct strategies in this stage. The first is classical *Large Neighborhood Search* (LNS) [31], which randomly selects a subset of variables for re-optimization. We further include *Adptive Large Neighborhood Search* (ALNS) [32] with adaptive neighborhood size, which dynamically adjusts the scope of variables based on search feedback. In addition, we provide two heuristics based on the integrality of the relaxed LP solution: the *Least Integral Heuristic* (LIH) [33], which focuses on variables farthest from integral values, and the *Most Integral Heuristic* (MIH) [34], which prioritizes those closest to integrality. Finally, the *Adaptive Constraint Partition* (ACP) method [35] leverages variable correlations to construct meaningful subproblems for focused refinement.

Table 1: Comparison of objective value results with baseline approaches using the same execution time. An upward arrow (↑) indicates that the result is better than the baseline. **Boldface** denotes the best result for each problem instance.

	SC_1	SC_2	MVC_1	MVC_2	MKS_1	MKS_2	MIS_1	MIS_2
Gurobi	24313.0	320036.5	27925.8	330816.4	34285.3	343707.2	-21966.7	-169223.2
SCIP	25317.5	919262.6	31256.7	490914.5	30616.0	1047136.9	-18687.9	-9125.2
Light-MILPopt	16528.1	164154.3	27548.1	278557.6	20589.2	208803.5	-22900.1	-228611.5
LEGO-Real (Ours)	**16108.6↑**	160693.4↑	**26675.4↑**	**271168.7↑**	**19957.7↑**	203306.6↑	-23085.9↑	-228792.6↑
LEGO-Gen (Ours)	16183.3↑	**160647.8↑**	26709.9↑	273948.6↑	20063.3↑	**203054.1↑**	**-23204.0↑**	**-230261.0↑**

Beyond local search, LEGO also allows the repaired prediction to guide commercial solvers directly. We integrate both *Gurobi* [27] and *SCIP* [28] as back-end solvers, enabling the predicted solution to serve as a warm-start or search bias. This dual-mode design—search-based refinement and solver-guided integration—makes this stage a flexible and powerful component in our framework.

4 Experiment

We conduct comprehensive experiments to evaluate the effectiveness, adaptability, and scalability of LEGO in solving mixed-integer linear programs (MILPs) through Predict-and-Search pipeline construction. The evaluation covers a diverse range of MILP problem settings, including both synthetic benchmarks and real-world instances. LEGO is compared against classical solvers and representative learning-based baselines, under standardized experimental protocols. The experimental settings are detailed in Section 4.1. To ensure a fair and comprehensive comparison, we employ multiple evaluation metrics to assess the performance of all methods considered in this study. Specifically, we include: a detailed comparison of solution quality under the same running time (Section 4.2), an evaluation of time efficiency under the same solution quality (Section 4.3), and a convergence analysis of the optimization process (Section 4.4).

4.1 Experimental Settings

Dataset. We consider four representative types of NP-hard MILP problems: Set Covering (SC) [36], Minimum Vertex Cover (MVC) [37], Maximum Independent Set (MIS) [38], and Mixed 0-1 Knapsack Set (MKS) [39]. For each problem type, we evaluate two representative scales: one with approximately 100K decision variables and constraints (e.g., MVC_1), and another with approximately 1M scale (e.g., MVC_2). All problems are formulated as minimization tasks. Detailed mathematical formulations, instance generation strategies, and exact instance sizes are provided in the appendix.

Baseline Approaches. We compare LEGO with three strong baselines: the state-of-the-art commercial solver *Gurobi* 12.0.1 [27], the academic open-source solver *SCIP* 9.2.1 [28], and the recent learning-based Predict-and-Search framework *Light-MILPopt* [26]. In addition, we evaluate two variants of LEGO. The first, *LEGO-Real*, uses access to large-scale training data for final size-transferable parameter tuning. The second, *LEGO-Gen*, assumes only small-scale training data is available and relies on our MILP-retrieval-based instance generator to synthesize large-scale instances for component selection and size-transferable parameter tuning. These two variants allow us to assess LEGO's performance under both data-rich and data-limited scenarios.

Environment. All experiments are conducted on a server equipped with Intel Xeon Platinum 8375C CPUs (2.90GHz) and four NVIDIA TESLA V100 GPUs (32GB each). Detailed experimental settings and runtime configurations are provided in the appendix.

4.2 Comparisons of Solution Effectiveness

To compare the solution effectiveness of different solvers, we evaluate all methods under the same execution time budget across eight large-scale MILP benchmarks, covering four problem types (SC, MVC, MKS, MIS) and two problem scales (100K and 1M). All problems are formulated as minimization tasks, where smaller objective values indicate better solution quality. This setting aligns with real-world scenarios, where solvers are often required to deliver high-quality solutions within fixed time limits. As shown in Table 1, our method LEGO achieves clearly better or comparable results across all tasks.

Table 2: Comparison of execution times under the same target value. A greater-than symbol (>) indicates the inability to achieve the target objective function in some instances within the maximum running time. **Boldface** is used to denote the best results.

	SC_1	SC_2	MVC_1	MVC_2	MKS_1	MKS_2	MIS_1	MIS_2
Gurobi	>30000s	>30000s	>30000s	>30000s	>30000s	>30000s	>30000s	>30000s
SCIP	>30000s	>30000s	>30000s	>30000s	>30000s	>30000s	>30000s	>30000s
Light-MILPopt	597.4s	3477.0s	594.7s	3473.4s	3909.9s	7931.2s	590.8s	3468.9s
LEGO-Real (Ours)	**228.4s↑**	1936.6s↑	103.8s↑	**536.0s↑**	2264.6s↑	**3285.1s↑**	**155.9s↑**	3367.7s↑
LEGO-Gen (Ours)	473.0s↑	**1542.1s↑**	**76.3s↑**	786.4s↑	**2038.3s↑**	4923.2s↑	225.0s↑	**2438.9s↑**

Traditional solvers such as Gurobi and SCIP yield significantly worse objective values under the same time constraints. For instance, Gurobi performs poorly on MVC_2 and MIS_2, while SCIP shows large suboptimality on SC_2 and MKS_2, where its returned values are more than five times worse than those of LEGO. These results indicate that classical solvers struggle to make effective progress on extremely large MILPs within limited time, due to the exponential search space and lack of learned heuristics. This contrast highlights the need for scalable methods in large-instance settings.

Compared to the strong learning-based baseline Light-MILPopt, both LEGO variants achieve consistently better objective values on all benchmarks. LEGO-Real performs particularly well on tasks where the training and test distributions are closely aligned, such as MVC_1 and MKS_1. LEGO-Gen, on the other hand, often outperforms LEGO-Real on large-scale or more complex tasks like SC_2 and MIS_2, thanks to its broader training distribution via synthetic MILP generation. These improvements demonstrate the benefit of LEGO's modular training strategy and its ability to generalize under different problem characteristics.

Overall, LEGO delivers strong and stable performance across diverse problem types and scales. Its hierarchical structure and component-wise learning enable it to adapt more effectively to large and complex MILPs than existing monolithic methods. The ability to outperform both traditional solvers and advanced learning-based baselines under the same time budget confirms LEGO's effectiveness as a general and practical framework for high-quality large-scale MILP solving.

4.3 Comparisons of Solving Efficiency

We evaluate the time efficiency of different methods under the same target objective value, meaning that all solvers are required to reach the same solution quality, and we compare the time needed to do so. As shown in Table 2, the two traditional solvers, Gurobi and SCIP, fail to reach the target within the 30,000-second limit on all tested instances. This highlights the difficulty of scaling general-purpose MILP solvers to high-dimensional problems with hundreds of thousands or millions of variables and constraints.

In contrast, both LEGO-Real and LEGO-Gen are able to reach the target solutions in significantly less time than the learning-based baseline Light-MILPopt. For example, on MVC_1, LEGO-Real reduces execution time from 594.7s to 103.8s (**82.5%** faster), while LEGO-Gen further reduces it to just 76.3s (**87.2%** faster). On SC_2, LEGO-Gen achieves a **55.6%** speedup over Light-MILPopt, while LEGO-Real saves **44.3%** of execution time. On average, LEGO reaches the same target values 2–6$\times$ faster across different problem types and scales. These results demonstrate that LEGO achieves comparable or better efficiency than the best existing learning-based Predict-and-Search framework.

We also observe that LEGO-Real is generally faster than LEGO-Gen on smaller-scale problems, likely due to its access to real large-instance training data, which helps specialize the learned policies. However, on larger-scale or more diverse instances (e.g., SC_2, MIS_2), LEGO-Gen often performs better, showcasing its stronger generalization capability through synthetic instance retrieval and adaptive training. This complementary behavior between the two variants suggests that LEGO is effective both when real data is available and when generalization to unseen distributions is required.

We attribute LEGO's superior efficiency to its flexible and adaptive design. By providing a rich set of modular components and a wide range of tunable hyperparameters, LEGO can adapt its optimization strategy to the structural characteristics of each specific MILP instance. This allows the framework to tailor its behavior more precisely, leading to faster convergence and better scalability. In contrast, Light-MILPopt follows a monolithic approach where a single learned policy is applied uniformly across all problems, limiting its ability to generalize or specialize to different tasks.

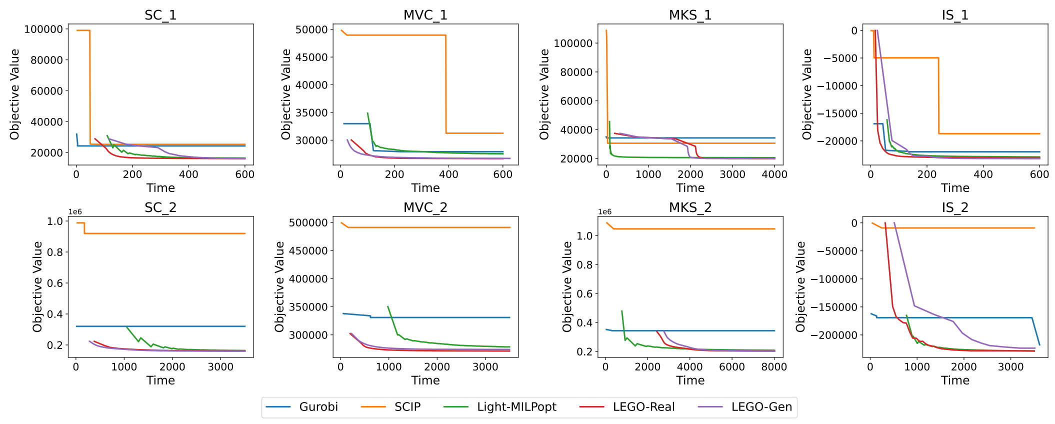

Figure 3: Time-objective convergence curves for all benchmark MILPs. Each subfigure shows the objective value (y-axis) over time (x-axis) for a specific MILP task.

LEGO's compositional nature thus enables more problem-aware optimization and consistently high performance across diverse benchmarks.

4.4 Analysis of Convergence

We analyze the convergence behavior of all methods by visualizing how the objective value evolves over time across eight benchmark MILPs, as shown in Figure 3. Each curve reflects how efficiently a solver improves solution quality under a fixed time budget. The results show that both LEGO-Real and LEGO-Gen converge significantly faster than traditional solvers (Gurobi, SCIP) and the learning-based baseline Light-MILPopt, especially in early stages of optimization.

LEGO variants consistently achieve rapid descent in the objective value curve, reaching high-quality solutions within a few minutes on 100K-scale problems, and under a few hundred seconds on 1M-scale ones. In contrast, Gurobi and SCIP either plateau early or make slow progress, particularly on large-scale instances like SC_2 and MIS_2. Light-MILPopt shows moderate convergence but lags behind LEGO in nearly all tasks. These patterns demonstrate the effectiveness of LEGO's structural decomposition and local decision policies, which enable faster and more focused optimization.

Moreover, LEGO-Gen often matches or even exceeds LEGO-Real in convergence speed on large-scale problems, such as MVC_2 and SC_2. This suggests that the synthetic training strategy and MILP retrieval mechanism empower LEGO-Gen with strong generalization and adaptation capabilities, even when real data is unavailable. Overall, the convergence curves highlight LEGO's robustness, scalability, and practical efficiency in solving large MILPs.

5 Conclusion

We propose **LEGO**, a general and modular framework for adaptive algorithm construction through component selection and configuration. Inspired by LEGO-style modularity, the framework is designed to assemble high-performing solving pipelines tailored to specific problem instances. In this work, we instantiate LEGO within the Predict-and-Search paradigm for solving large-scale MILPs, where it selects and configures components and hyperparameters based on structural priors. To enhance adaptability, LEGO can generate synthetic training data at varying scales and leverage hybrid evaluation metrics, including LLM-informed assessments, to identify robust configurations even under limited training dataset.

Extensive experiments show that LEGO outperforms traditional solvers and strong learning-based baselines in both convergence speed and solution effectiveness. However, our current framework relies on the quality of the problem generator for training instance construction. In future work, we aim to further enhance the task-driven evaluation metrics to better align with practical objectives, and continuously expand the component library by integrating state-of-the-art modules, thereby improving LEGO's adaptability and performance across diverse problem settings.

References

[1] Stefan Bock, Stefan Bomsdorf, Nils Boysen, and Michael Schneider. A survey on the traveling salesman problem and its variants in a warehousing context. *European Journal of Operational Research*, 2025.

[2] Meng-Yu Huang, Ling-Ying Huang, Yu-Xing Zhong, Hui-Wen Yang, Xiao-Meng Chen, Wei Huo, Jia-Zheng Wang, Fan Zhang, Bo Bai, and Ling Shi. Milp acceleration: A survey from perspectives of simplex initialization and learning-based branch and bound. *Journal of the operations research society of china*, 2025.

[3] Phuong Le. A survey on combinatorial optimization. *arXiv preprint arXiv:2409.00075*, 2024.

[4] Per Kristian Lehre and Shishen Lin. No free lunch theorem and black-box complexity analysis for adversarial optimisation. *Advances in Neural Information Processing Systems*, 2024.

[5] Xiaoqian Li and Kwan L Yeung. Traffic engineering in segment routing networks using milp. *IEEE Transactions on Network and Service Management*, 2020.

[6] Carlos A Mendez and Jaime Cerdá. An milp framework for batch reactive scheduling with limited discrete resources. *Computers & Chemical Engineering*, 2004.

[7] Raine Jokinen, Frank Pettersson, and Henrik Saxén. An milp model for optimization of a small-scale lng supply chain along a coastline. *Applied Energy*, 2015.

[8] Qingyu Han, Linxin Yang, Qian Chen, Xiang Zhou, Dong Zhang, Akang Wang, Ruoyu Sun, and Xiaodong Luo. A gnn-guided predict-and-search framework for mixed-integer linear programming. In *The International Conference on Learning Representations*, 2023.

[9] Taoan Huang, Aaron M Ferber, Arman Zharmagambetov, Yuandong Tian, and Bistra Dilkina. Contrastive predict-and-search for mixed integer linear programs. In *International Conference on Machine Learning*, 2024.

[10] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: An automatic algorithm configuration framework. *Journal of artificial intelligence research*, 2009.

[11] Gellért Weisz, Andras Gyorgy, and Csaba Szepesvári. Leapsandbounds: A method for approximately optimal algorithm configuration. In *International Conference on Machine Learning*, 2018.

[12] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. In *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach*. 2016.

[13] Zeyuan Ma, Hongshu Guo, Yue-Jiao Gong, Jun Zhang, and Kay Chen Tan. Toward automated algorithm design: A survey and practical guide to meta-black-box-optimization. *arXiv preprint arXiv:2411.00625*, 2024.

[14] Massimo Regona, Tan Yigitcanlar, Bo Xia, and Rita Yi Man Li. Opportunities and adoption challenges of ai in the construction industry: A prisma review. *Journal of open innovation: technology, market, and complexity*, 2022.

[15] Marcus Tantakoun, Xiaodan Zhu, and Christian Muise. Llms as planning modelers: A survey for leveraging large language models to construct automated planning models. *arXiv preprint arXiv:2503.18971*, 2025.

[16] Frank Hutter, Lars Kotthoff, and Marius Lindauer. Algorithm configuration: Challenges, methods and perspectives. Tutorial at IJCAI 2020, 2020. Accessed: 2025-05-11.

[17] Tobias Achterberg. Constraint integer programming. 2007.

[18] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.

[19] Jiayi Zhang, Chang Liu, Xijun Li, Hui-Ling Zhen, Mingxuan Yuan, Yawen Li, and Junchi Yan. A survey for solving mixed integer programming via machine learning. *Neurocomputing*, 2023.

[20] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[21] Tianxing Yang, Huigen Ye, and Hua Xu. Code retrieval for milp instance generation. *arXiv preprint*, 2025.

[22] 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*, 2019.

[23] Jian-Ya Ding, Chao Zhang, Lei Shen, Shengyin Li, Bing Wang, Yinghui Xu, and Le Song. Accelerating primal solution findings for mixed integer programs based on solution prediction. In *AAAI Conference on Artificial Intelligence*, 2020.

[24] Ziang Chen, Jialin Liu, Xinshang Wang, and Wotao Yin. On representing mixed-integer linear programs by graph neural networks. In *International Conference on Learning Representations*, 2023.

[25] Nicolas Sonnerat, Pengming Wang, Ira Ktena, Sergey Bartunov, and Vinod Nair. Learning a large neighborhood search algorithm for mixed integer programs. *arXiv preprint arXiv:2107.10201*, 2021.

[26] Huigen Ye, Hua Xu, and Hongyan Wang. Light-milpopt: Solving large-scale mixed integer linear programs with lightweight optimizer and small-scale training dataset. In *International Conference on Learning Representations*, 2024.

[27] Tobias Achterberg. What's new in gurobi 9.0. *Webinar Talk url: https://www. gurobi. com/wp-content/uploads/2019/12/Gurobi-90-Overview-Webinar-Slides-1.pdf*, 2019.

[28] Tobias Achterberg. Scip: Solving constraint integer programs. *Mathematical Programming Computation*, 2009.

[29] Taehyun Yoon. Confidence threshold neural diving. *arXiv preprint arXiv:2202.07506*, 2022.

[30] Huigen Ye, Hua Xu, Hongyan Wang, Chengming Wang, and Yu Jiang. Gnn & gbdt-guided fast optimizing framework for large-scale integer programming. In *International Conference on Machine Learning*, 2023.

[31] Jialin Song, Yisong Yue, Bistra Dilkina, et al. A general large neighborhood search framework for solving integer linear programs. *Advances in Neural Information Processing Systems*, 2020.

[32] Gregor Hendel. Adaptive large neighborhood search for mixed integer programming. *Mathematical Programming Computation*, 2022.

[33] Vinod Nair, Mohammad Alizadeh, et al. Neural large neighborhood search. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020.

[34] Timo Berthold. *Primal heuristics for mixed integer programs*. PhD thesis, Zuse Institute Berlin (ZIB), 2006.

[35] Huigen Ye, Hongyan Wang, Hua Xu, Chengming Wang, and Yu Jiang. Adaptive constraint partition based optimization framework for large-scale integer linear programming. In *AAAI Conference on Artificial Intelligence*, 2023.

[36] Egon Balas and Manfred W Padberg. On the set-covering problem. *Operations Research*, 1972.

[37] Samir Khuller. Algorithms column: The tertex cover problem. *ACM SIGACT News*, 2002.

[38] Jan-Henrik Haunert and Alexander Wolff. Beyond maximum independent set: An extended integer programming formulation for point labeling. *ISPRS International Journal of Geo-Information*, 2017.

[39] Alper Atamtürk. Cover and pack inequalities for (mixed) integer programming. *Annals of Operations Research*, 2005.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and the final paragraph of the introduction clearly state the paper's contributions, including the formulation of LEGO as a general framework for adaptive algorithm construction and its instantiation in MILP solving. These claims are consistent with the theoretical formulation and empirical results presented in the paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations of our approach are discussed at the end of the Conclusion section, where we acknowledge the framework's reliance on the quality of the problem generator and outline future work to address this limitation by developing more generalizable generation and retrieval mechanisms.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: The paper does not include formal theoretical results or proofs; it focuses on a modular framework and its empirical evaluation.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide detailed descriptions of experimental configurations, solver settings, and hyperparameter choices in both the main text (Section 4) and Appendix A. Additionally, we have released a demo of LEGO at `https://anonymous.4open.science/r/LEGO-B36D`, which includes code and instructions to facilitate quick reproduction of our key results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide open access to a demo implementation of LEGO at `https://anonymous.4open.science/r/LEGO-B36D`, which includes code, example data, and detailed instructions to reproduce the main experimental results. Further reproduction details are also described in the supplemental material, following NeurIPS code and data submission guidelines.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide comprehensive descriptions of the experimental setup in both the main paper (Section 4) and the appendix, including details on datasets, data splits, hyperparameter settings, and selection strategies. In addition, the open-source demo at `https://anonymous.4open.science/r/LEGO-B36D` contains the full codebase and configuration files needed to reproduce the experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We provide error bar analyses based on multiple experimental runs in the appendix, including explanations of the sources of variability (e.g., random seeds) and the method used for computing standard deviations. These results support the robustness and statistical significance of our main empirical findings.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We specify the type of compute workers used (e.g., CPU/GPU configurations) in the experimental setup section of the main paper. An estimate of the total compute cost, including time and resource usage across different experiments, is provided in the appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have carefully reviewed the NeurIPS Code of Ethics and confirm that our research complies fully with its principles. Our work does not involve human subjects, private or sensitive data, or applications with foreseeable harmful impact. We have followed best practices in dataset usage, code release, and reproducibility, and have taken care to consider potential societal and environmental impacts.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: Our work focuses on the development of a general-purpose optimization framework and does not involve data generation, interaction with end-users, or deployment in application-specific contexts. As such, we do not foresee any direct societal impact—positive or negative—at this stage, nor any obvious potential for malicious use.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA] .

    Justification: Our work does not involve the release of pretrained generative models, large language models, or datasets that carry a high risk of misuse. Therefore, no specific safeguards are necessary.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All external assets used in our work, including datasets, codebases, and models, are properly cited in the main paper. We have ensured that their licenses and terms of use are respected, and we include references to the original papers and repositories where applicable.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Our paper introduces new algorithmic components, which we plan to open-source upon publication. A demo showcasing core functionalities is already publicly available and can be freely used. The final release will include detailed documentation covering usage instructions, license, limitations, and implementation details.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Our work does not involve any form of crowdsourcing or research with human subjects.

    Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA] .

    Justification: Our research does not involve human subjects or any form of human-subject experimentation, and therefore no IRB or equivalent ethical review was necessary.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: Our paper includes an LLM-enhanced performance evaluation component, where large language models are used to generate novel evaluation metrics. The specific models used, their configurations, and their role in the evaluation process are clearly described in the paper, along with appropriate citations.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.