THEORY-DRIVEN MODELING AND LLM-GUIDED EVOLUTION FOR POWER SYSTEM SCHEDULING

Wushuaijun Zhang*
School of Science
Beijing University of Posts and Telecommunications
Beijing 100876, China
{zhangwsj2024}@bupt.edu.cn

Yikun Zong*
Department of Engineering
University of Cambridge
Cambridge, CB3 0DG, United Kingdom
{yz977}@cam.ac.uk

Yishuang Yue
Academy of Mathematics and Systems Science
Chinese Academy of Sciences
Beijing 100190, China
{yueyishuang}@amss.ac.cn

Kun Yuan†
Centre for Machine Learning Research
Peking University
Beijing, 100871, China
{kunyuan}@pku.edu.cn

ABSTRACT

Unit Commitment (UC) and Economic Dispatch (ED) are critical mixed-integer programming (MIP) formulations in modern power systems. The effectiveness of warm-start strategies for large-scale UC/ED MIPs crucially depends on variable selection and hyperparameter configuration. However, existing approaches either rely on expert-designed heuristics or computationally expensive machine learning methods, both of which struggle to adapt effectively across diverse problem structures. To address this, we present a *recursive self-improvement* framework where an AI system continuously refines its own decision-making strategies through LLM-guided evolutionary learning. The system diagnoses its performance failures, critiques its scoring functions, and autonomously updates all parameters through iterative evolution based on fitness feedback. This recursive loop enables the system to discover interpretable scoring functions (e.g., log-transformed reduced costs) and optimal hyperparameters without manual intervention, demonstrating measurable, reliable, and deployable self-improvement in industrial optimization settings. Experimental results on realistic UC/ED benchmarks demonstrate that our approach significantly outperforms commercial solvers such as Gurobi and baseline methods, achieving substantial speedup while maintaining solution quality. The framework eliminates manual parameter tuning and enables automatic adaptation to diverse problem structures. Our work is available at https://anonymous.4open.science/r/Power_L2O_evolve-EFF4/.

1 INTRODUCTION

Optimization is central to large-scale energy systems, where Unit Commitment (UC) and Economic Dispatch (ED) form the backbone of day-ahead scheduling (Wood et al., 2013). These problems are modeled as Mixed-Integer Programs (MIPs) with thousands of binary on/off variables and complex temporal constraints such as minimum up/down times and ramping (Garver, 2008). Although state-of-the-art solvers (e.g., Gurobi, HiGHS) are highly optimized, solving industrial-scale UC/ED instances remains challenging, with many instances failing to reach acceptable optimality gaps within practical time limits.

These formulations almost invariably adopt convex piecewise-linear (PWL) cost models for generation, which remain the industry standard in ISO/RTO market submissions (Vielma et al., 2010).

*These authors contributed equally to this work.
†Corresponding author.

However, standard convex-combination PWL formulations introduce numerous auxiliary variables and constraints, inflating problem size and weakening LP relaxations. Traditional modeling approaches often contain significant redundancy, while traditional learning methods such as Learning-to-Optimize (L2O) (Gasse et al., 2019) suffer from poor generalization across diverse problem structures. This dual limitation motivates our two-pronged acceleration strategy: **data cleaning and model presolve** to eliminate redundancy and improve problem structure, and **learning-driven acceleration** (Novikov et al., 2025) to discover effective solving heuristics with strong generalization.

We address this challenge through a recursive self-improvement framework that combines *data cleaning and model presolve* with *LLM-guided evolution*. The data cleaning stage automatically reformulates PWL cost functions into sparse semi-continuous formulations, eliminating redundant constraints and reducing problem dimensionality. The LLM-guided evolution stage enables the system to automatically discover effective scoring functions and hyperparameters through iterative refinement based on fitness feedback. This combination enables dramatic speedup performance in experiments and our key innovations are as follows:

Recursive Self-Improvement Perspective Our framework embodies recursive self-improvement (RSI) principles through a three-step cycle: (1) *self-diagnosis*: evaluating fitness to identify weaknesses, (2) *self-critique*: LLMs analyze failures and generate improvement suggestions, and (3) *self-update with selective adoption*: LLMs serve as intelligent mutation operators to generate improved configurations, retaining only those that improve performance. The system learns how to score variables through recursive refinement, evolving from complex multi-feature functions to simple, effective forms (e.g., log-transformed reduced costs).

- **Recursive Self-Improvement Framework for Scoring Function Discovery.** We propose a recursive self-improvement framework where the system automatically discovers interpretable scoring functions (e.g., log-transformed reduced costs) and optimal hyperparameters through iterative refinement based on fitness feedback, without manual intervention. By *interpretable* we mean the evolved form is a simple closed-form expression with clear LP interpretation; by *discover* we mean the evolution converges to this form from a richer initial search space.
- **Significant Performance Improvements.** We demonstrate substantial improvements on realistic piecewise linear UC/ED dispatch instances: while commercial solvers fail to reach 3% MIP gap within 1800 seconds, our approach achieves this threshold in approximately 30 seconds on average, representing a 60$\times$ speedup while maintaining solution quality.
- **Open-Source Benchmark for Power System MIP.** Existing power system benchmarks are limited in scale (typically 10–100 units) and lack comprehensive physical parameters. We create and open-source a large-scale benchmark dataset (10–600 units) with comprehensive physical parameters, generated from real-world instances (MIIT Energy Electronics Innovation Competition scenarios and national power system reports, calibrated against FERC and PJM market profiles) and made publicly available for evaluating MIP solvers on power system scheduling problems.

2 Related Work

Classical Optimization and PWL Reformulation. UC/ED problems have long been studied as separable convex MIP formulations. Early works addressed convex function minimization (Charnes & Lemke, 1954; Dantzig et al., 1958), with integer programming models proposed in (Garver, 2008). More recent work refines PWL cost modeling with tight MIP formulations (Vielma et al., 2010; Morales-España et al., 2015; Carrión & Arroyo, 2006), and these models remain dominant in power system operations (Wood et al., 2013). Semi-continuous reformulations have been studied as alternatives to SOS2 formulations (Cost, 2003), offering improved sparsity and numerical conditioning. Our work extends this line by developing an automatic detection and rewriting pipeline that applies semi-continuous reformulation with interval reasoning for bound tightening, providing a principled foundation for acceleration.

Presolve and Model Reduction. Presolve techniques aim to reduce problem size and improve numerical properties before solving. Classical presolve methods include constraint elimination,

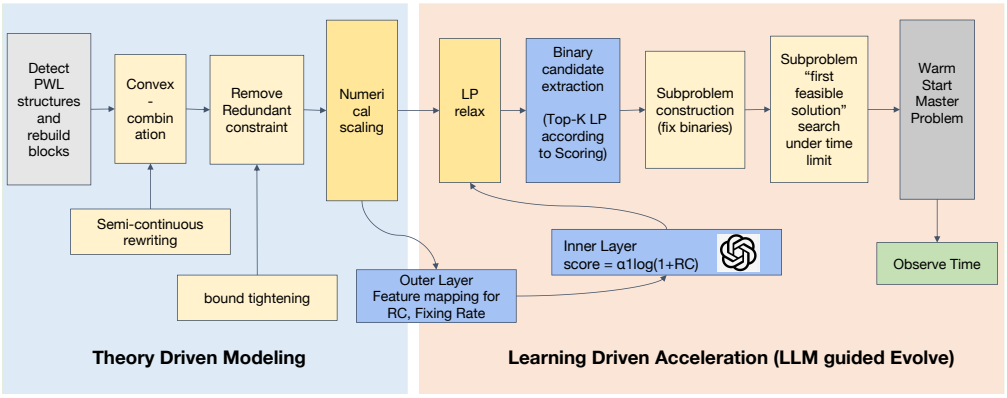


Figure 1: Overview of the proposed **LLM+Evolve** pipeline. The process starts from presolve and LP relaxation, extracts binary candidates, applies LLM-guided scoring and fixing, constructs subproblems, obtains a feasible solution, warm-starts the master solver, and finally refines the configuration via evolutionary search.

variable fixing, and coefficient tightening (Andersen & Andersen, 1995). Domain-specific presolve for UC/ED has focused on constraint aggregation and temporal decomposition (Frangioni & Gentile, 2006). Our approach differs by focusing on PWL reformulation and interval-based bound tightening, which directly addresses the computational bottleneck of convex-combination formulations.

Learning-to-Optimize (L2O) and Algorithm Configuration. L2O methods aim to learn optimization strategies from data. (Gasse et al., 2019) use graph convolutional neural networks for exact combinatorial optimization, learning branching strategies for mixed-integer programs. (Ding et al., 2020) accelerate primal solution findings for MIPs based on solution prediction, effectively learning warm-start heuristics. Recent theoretical advances have improved generalization guarantees (Sucker & Ochs, 2024), meta-tuning frameworks (Sharifnassab et al., 2024; Pan et al., 2025), and algorithmic robustness (Castera & Ochs, 2024; Chen et al., 2020). However, many of these models rely on extensive supervised training and lack interpretability. Algorithm configuration methods typically require manual parameter tuning or expensive search over configuration spaces. Our approach differs by using supervised learning specifically for algorithm configuration, learning from evolution-discovered optimal hyperparameters rather than end-to-end training, enabling instance-specific adaptation with minimal training data.

DRL and GNN-based MIP Solvers. Deep reinforcement learning and GNNs have been widely applied to combinatorial optimization. (Li et al., 2021) address TSP variants with DRL; (Zhang et al., 2022) learn branching strategies; (Sorokin & Kostin, 2023) minimize B&B tree depth; and MIP-GNN (Khalil et al., 2022) offers a GNN-based framework for MIP decision guidance. These models are often tightly coupled to specific problem structures and require heavy data or simulation environments for training. In contrast, our learning-driven acceleration approach requires minimal training data and discovers interpretable heuristics.

LLMs for Optimization and Evolutionary Search. Large language models have recently been applied in optimization via code generation and heuristic evolution (Ye et al., 2024; Xie et al., 2025), typically focusing on generating code or full solutions. OpenEvolve and reflective strategies demonstrate LLM effectiveness beyond text tasks. Evolutionary computation methods such as CMA-ES or differential evolution have long been used for heuristic parameter tuning. Recent ML-hybrid methods (e.g., (Lange et al., 2025; Chauhan et al., 2025)) combine learned guidance with low-cost perturbation, typically using traditional mutation operators or learned policies. Our approach extends this paradigm by using LLMs as intelligent mutation operators within the evolutionary loop, enabling automatic discovery of interpretable scoring functions for MIP variable selection.

3 METHODOLOGY

3.1 PROBLEM FORMULATION AND DATA MODELING

Joint UC/ED Framework We formulate the joint Unit Commitment and Economic Dispatch (UC/ED) problem as a Mixed-Integer Linear Program (MILP) to minimize operating costs over a multi-period scheduling horizon ($T = 24$). This model optimizes both binary commitment and continuous dispatch decisions subject to practical system and unit-level constraints (Morales-España et al., 2015; Wood et al., 2013; Carrión & Arroyo, 2006). The resulting formulation provides a realistic and challenging testbed for evaluating the proposed heuristic discovery method.

Dataset and Benchmark While classic UC/ED benchmarks (e.g., IEEE 10-unit, Kazarlis 10-unit) provide valuable academic baselines, they are limited in scale (typically 10–100 units) and cannot adequately evaluate modern MIP solvers on industrial-scale problems (300+ units). Furthermore, traditional benchmarks often lack comprehensive physical parameters such as detailed ramp rates, minimum up/down times, startup costs, and realistic piecewise-linear fuel cost functions with multiple breakpoints. To address these limitations, we create a scalable benchmark extending from 10 to 600 units, maintaining academic consistency while providing the scale necessary for realistic performance assessment. Unit parameters and cost coefficients are calibrated against FERC (Commission et al., 2015) and PJM (Monitoring Analytics, 2015) market profiles. Fuel cost breakpoints and generator fleet parameters are derived from anonymized, high-resolution scenario data provided by the *MIIT Energy Electronics Innovation Competition (Key Information Technology Track)*, reflecting provincial-level power system dispatch conditions with weekday demand distinctions and forced outage perturbations. These design choices ensure the benchmark instances are both academically standard and closely aligned with real-world UC/ED practices. The full dataset is available at this anonymous link https://github.com/PowerSystem-Evolution/llmfiles.git

The formulation captures key operational features: load balance matches total generation to demand, generation limits bound output based on unit status, ramping limits restrict inter-period output changes, power adequacy ensures committed capacity covers demand, startup/shutdown logic tracks status transitions, and piecewise-linear (PWL) fuel cost constraints approximate nonlinear cost curves. A complete formulation with objective, constraints, and notation is provided in Appendix A.

3.2 DATA CLEANING AND MODEL PRESOLVE

Figure 1 illustrates the overall pipeline. Before applying LLM-guided evolution, we perform data cleaning and model presolve to eliminate redundancy and improve problem structure. Direct UC/ED formulations with convex-combination PWL costs are computationally expensive: each generator with K_i cost segments introduces K_i convexity variables and adjacency constraints, inflating problem size and weakening LP relaxations.

Semi-Continuous PWL Reformulation: We replace the standard convex-combination representation with semi-continuous multiple-choice formulation (see Algorithm 3 in appendix for details). This reformulation eliminates convexity and adjacency constraints, yields a smaller and better-scaled constraint matrix, and preserves the same feasible (P, F) space and optimal objective as the ideal SOS2 representation (Cost, 2003), while offering superior sparsity and numerical conditioning for large-scale UC/ED instances.

Automatic Detection and Rewriting: We automatically detect PWL blocks through row classification (set-partitioning, cumulative, bound-link, convex-combination rows), discover semi-continuous variables, reconstruct ordered PWL blocks, and rewrite the model by replacing λ-based convex-combination with semi-continuous variables $w^k_{i,t}$. Interval-based bound tightening derives tighter variable bounds from constraint coupling, reducing feasible regions and improving solver performance; bound tightening and coefficient scaling further improve numerical conditioning.

This data cleaning and model presolve stage significantly reduces variable and constraint counts, tightens LP relaxations, and improves numerical stability, producing compact instances that serve as the entry point for the subsequent LLM-guided evolution pipeline. The complete automated data

cleaning and model presolve process is detailed in Algorithm 3 in the appendix. The mathematical equivalence between the convex-combination and semi-continuous formulations is formally proven in Appendix D.

3.3 LLM-GUIDED EVOLUTION FOR AUTOMATED HEURISTIC DISCOVERY

Our pipeline employs a *recursive self-improvement framework* where all parameters are automatically discovered through LLM-guided evolution. This design enables automatic adaptation to diverse problem instances without manual parameter tuning.

3.3.1 RECURSIVE SELF-IMPROVEMENT VIA LLM-GUIDED EVOLUTION

We implement *recursive self-improvement* through LLM-guided evolution (see Algorithm 5 in appendix for complete pseudocode) where the system continuously refines all parameters based on fitness feedback. Each configuration in the population includes scoring function parameters $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$, reduced cost (RC) threshold $\hat{\theta}_{\text{RC}}$ (variables with $RC > \hat{\theta}_{\text{RC}}$ are fixed to 0; see Algorithm 4) from $\{100K, 300K, 500K, 700K, 1000K\}$, and fixing rate $\hat{\theta}_{\text{FR}}$ (fraction of top-scored variables to fix, $k = \lfloor \hat{\theta}_{\text{FR}} \times |\mathcal{C}| \rfloor$) in $[0.4, 0.6]$. LLMs analyze 11 structural features (problem scale, variable types, constraint structures) extracted from the MPS (Mathematical Programming System format) file to generate configurations. The core innovation is that the system *learns how to score variables and configure all hyperparameters* through iterative self-improvement, evolving from complex multi-feature scoring functions to simple, effective forms.

Scoring Function and Population Initialization. The algorithm starts with a multi-feature scoring function $\text{score}(x) = \alpha_1 \cdot f_1(RC) + \alpha_2 \cdot \text{near01} + \alpha_3 \cdot \text{period\_bonus} - \alpha_4 \cdot \text{conflict\_risk}$, where $\text{near01} = 0.5 - \min(LP, 1 - LP)$ measures proximity to integrality, period_bonus rewards variables in critical time periods (e.g., peak demand), and conflict_risk penalizes variables that may violate temporal constraints (e.g., min up/down) when fixed. The population is initialized with LLM-generated seeds, diverse nonlinear functions, and Gaussian perturbations.

Evolution Loop and Fitness Evaluation. For each generation $g = 1$ to $gens$, the system performs the following recursive self-improvement cycle:

Step 1: Self-Diagnosis - Each configuration is evaluated end-to-end: compile scoring function, rank variables, solve subproblem, warm-start master, and compute fitness.

Fitness Function Design. The fitness employs a hierarchical strategy prioritizing solution quality over runtime. For gap-satisfied configurations (final MIP gap ≤ 0.001), fitness uses efficiency score $\log(1 + \text{runtime}) + 0.01 \cdot \log(1 + \text{node\_count})$ (negated, larger is better). For gap-unsatisfied configurations, fitness uses penalty $1000 + 40 \cdot \text{rel\_init\_quality} + 60 \cdot \text{gap}$ where $\text{rel\_init\_quality} = |\text{init\_solution} - \text{dual\_bound}|/(|\text{dual\_bound}| + 1)$ measures warm-start quality. The base offset (1000) ensures gap-unsatisfied configurations receive lower fitness than any gap-satisfied configuration. This hierarchical design ensures the process first achieves gap thresholds, then optimizes efficiency.

Step 2: Self-Critique - Top-μ configurations (where $\mu \in \mathbb{N}$ is the number of parent configurations) are selected as parents. LLMs analyze performance feedback (best objective, improvement, feasibility, iteration count, trends) and generate critiques identifying weaknesses (e.g., "scoring function overweights LP proximity, causing infeasible subproblems" or "reduced cost should be emphasized more strongly").

Step 3: Self-Update via LLM-Guided Mutation and Selective Adoption. LLMs serve as intelligent mutation operators, generating improved configurations through LLM-MUTATE (see Algorithm 5 in appendix for details). The LLM receives comprehensive context: parent configuration (scoring parameters $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ and hyperparameters RC threshold, fixing rate), performance feedback (fitness, gap, runtime, feasibility), and stage-specific constraints. In linear stage (iteration < 800), only α_1 in $\alpha_1 \cdot \log(|RC| + 1)$ is evolved; in nonlinear stage (≥ 800), LLMs can update all weights or propose new scoring function forms. Hyperparameters are adjusted based on performance feedback: e.g., reducing fixing rate if subproblems are frequently infeasible, adjusting RC threshold if gap is too large. Unlike random Gaussian perturbations, LLM-guided mutations are

Algorithm 1 Recursive Self-Improvement Loop for Scoring Functions

Input: $\mathcal{P}_0$, G, $\tilde{\mathcal{M}}$ **Output:** θ^*
for $g = 1$ to G **do**
  $\mathcal{S} \leftarrow \{(\theta_i, \textsc{EvaluateFitness}(\theta_i)) : \theta_i \in \mathcal{P}_{g-1}\}$
  $\mathcal{P}_{\text{parents}} \leftarrow \textsc{TopMu}(\mathcal{S}, \mu)$; $\mathcal{C} \leftarrow \textsc{AnalyzeFailures}(\mathcal{P}_{g-1}, \mathcal{S})$
  $\mathcal{P}_g \leftarrow \mathcal{P}_{\text{parents}}$
  while $|\mathcal{P}_g| < |\mathcal{P}_0|$ **do**
    $\mathcal{P}_g \leftarrow \mathcal{P}_g \cup \{\textsc{LLMMutate}(\text{random}(\mathcal{P}_{\text{parents}}), \mathcal{C})\}$
  end while
  $\mathcal{P}_g \leftarrow \{\theta \in \mathcal{P}_g : f(\theta) \geq f(\theta_{\text{parent}})\}$
end for
return $\arg\max_{\theta \in \mathcal{P}_G} f(\theta)$

context-aware and leverage domain knowledge embedded in the language model. Only configurations that improve fitness are retained, ensuring monotonic improvement across generations.

Automatic Discovery of Simplified Scoring Functions. Through the recursive self-improvement loop, the system automatically discovers that reduced cost is the dominant feature, evolving from complex multi-feature functions to $\text{score}(x) = \alpha_1 \cdot \log(|RC(x)| + 1)$. The complete process is detailed in Algorithm 4 in the appendix. This recursive self-improvement framework enables the system to learn how to score variables and configure hyperparameters through iterative evolution, with both components evolving together based on fitness feedback. The system consistently converges to reduced-cost-based scoring and optimal hyperparameters, demonstrating measurable and deployable self-improvement in industrial optimization.

3.3.2 Variable Ranking and Subproblem Construction

The pipeline constructs subproblems using the evolved configuration: the evolved scoring function (typically $\alpha_1 \cdot \log(|RC| + 1)$) and hyperparameters (RC threshold and fixing rate) are all obtained through LLM-guided evolution. Algorithm 4 details the complete process: after solving the LP relaxation, the evolved scoring function ranks all candidate variables, and the top-k variables (where $k = \lfloor \hat{\theta}_{\text{FR}} \times |\mathcal{C}| \rfloor$) are fixed based on the evolved RC threshold ($RC > \hat{\theta}_{\text{RC}}$ fixes to 0, $RC < -10$ fixes to 1, $|RC| < 10^{-6}$ fixes based on LP value; this threshold was found optimal through evolution) and fixing rate. After constraint propagation, the restricted subproblem is solved to find a feasible solution for warm-starting the master problem.

3.3.3 Warm Start of the Master Problem

The feasible solution from the subproblem is immediately injected into the master problem via Gurobi's `Start` attribute, providing an initial incumbent and upper bound that enables aggressive pruning and accelerates convergence. The master problem is solved with a time limit (typically 1800 seconds) and a target MIP gap of 3%, with solution time measured to evaluate convergence speed.

This recursive self-improvement framework represents a novel approach to *automated heuristic design for optimization*, where all parameters (hyperparameters and scoring function) are evolved together automatically. The observation that LLM-guided evolution converges to RC-based scoring validates both the importance of reduced cost in MILP variable selection and the effectiveness of automated heuristic discovery for industrial optimization problems

4 Experiments

4.1 Setup and Baseline

We benchmark our recursive self-improvement framework on realistic UC/ED instances using Gurobi v10.0. We evaluate two types of instances: (1) **normal instances**, representing standard UC/ED scenarios, and (2) **disturbed instances**, where key parameters (electricity prices and ramping limits) have been perturbed to assess robustness. Performance is measured using two comple-

mentary metrics: (1) **runtime (s)** to achieve a 3% MIP gap. All reported results are averaged across multiple runs, and (2) **optimality gap (%)** after a fixed time limit (1200s).

The **baseline** is Gurobi directly solving the original models without data cleaning or our recursive self-improvement framework. The baseline fails to reach the 3% MIP gap threshold within 1800 seconds for both normal and disturbed instances, effectively rendering the problems *unsolvable* under practical time constraints. Evolution is performed on a small subset of instances for approximately 3 hours; the resulting configurations generalize to the full benchmark, demonstrating minimal data requirements. Our method applies the framework to the same dataset (both normal and disturbed).

Table 1: Mean MIP Gap Comparison: Gap After 1200s Time Limit (%)

Dataset	**Gurobi**	**Modeling**	**Modeling with Learning**
Models	4.30	1.80	0.75

For complete gap comparison across all instances, see Table 6 (Table 7) in Section H.

Table 2: Performance Improvements: Time to Achieve 3% Gap (seconds) for Normal Dataset

Part 1				**Part 2**			
Model ID	**Gurobi**	**Presolve**	**After Evolve**	**Model ID**	**Gurobi**	**Presolve**	**After Evolve**
67	1800.00	4.00	1.33	70	1800.00	8.49	2.96
64	1800.00	5.31	2.25	298	1800.00	80.20	30.90
73	1800.00	6.15	2.68	85	1800.00	12.45	5.12
76	1800.00	7.22	3.15	88	1800.00	15.32	6.28
79	1800.00	9.18	4.05	91	1800.00	18.67	7.89
82	1800.00	11.25	5.12	94	1800.00	22.15	9.45
103	1800.00	14.32	6.58	112	1800.00	28.45	12.15
106	1800.00	16.78	7.89	118	1800.00	32.15	13.68
109	1800.00	19.45	9.12	124	1800.00	36.78	15.89
115	1800.00	23.67	11.25	130	1800.00	41.25	18.45
121	1800.00	27.89	13.45	136	1800.00	45.67	20.12
127	1800.00	31.45	15.23	142	1800.00	49.23	22.45
133	1800.00	35.67	17.89	148	1800.00	53.45	24.67
139	1800.00	39.23	19.56	154	1800.00	57.89	26.89
145	1800.00	43.45	21.23	160	1800.00	62.15	28.45
				Mean	**1800**	**75.13**	**30.96**

For complete runtime results across all instances, see Table 4 in Section G.

4.2 Analysis

- **Solution quality improvement:** As shown in Table 1, our method achieves significantly lower gaps than the baseline, with a mean MIP gap of 0.75% after 1200 seconds, compared to 4.30% for the baseline, representing a **5.7× improvement** in solution precision. This demonstrates that our approach not only accelerates convergence but also significantly enhances solution quality.
- **Acceleration solving efficiency:** Table 2 reveals the dramatic speedup achieved by our framework. When targeting a 3% MIP gap, Gurobi on the original model fails to reach this threshold within 1800 seconds (30 minutes), effectively rendering the problem *unsolvable* under practical time constraints. Our data cleaning and model presolve step transforms these instances into solvable problems, with Gurobi on presolved models achieving 3% gap in approximately 75 seconds, representing a **24× speedup** over the original formulation. The addition of our learning-driven acceleration framework further accelerates convergence to 30 seconds, achieving a **60× overall speedup** while maintaining solution quality. This combination of modeling and learned heuristics enables practical real-time optimization for industrial UC/ED scheduling.
- **Speedup Mechanisms:** The performance gains come from two complementary mechanisms. **Data cleaning and model presolve** eliminates redundant variables and constraints

through semi-continuous reformulation, tightening LP relaxations and reducing problem dimensionality. **Learning-driven acceleration** provides high-quality warm starts through heuristic from LLM-guided evolution that discovers effective scoring functions, combined with machine learning that predicts instance-specific hyperparameters (RC threshold and fixing rate) from MPS features, providing superior upper bounds for faster convergence.

- **Ablation Analysis:** Comparing baseline, modeling-only, and the full framework (modeling with learning heuristic), modeling alone achieves a **24× speedup**, while learning-driven acceleration contributes an additional **2.6× speedup**, resulting in a **60× overall speedup**. Both components are essential: modeling improves problem structure, while learning heuristic optimizes variable selection and warm-start generation.

4.3 Robustness: Perturbation of Price and Ramping

To evaluate robustness, we perform ablation studies by perturbing two key physical parameters of the original UC/ED instances: (1) **electricity price (fuel cost)**: we apply multiplicative noise $\beta_i^k \leftarrow \beta_i^k \cdot (1+\epsilon)$ where $\epsilon \sim \mathcal{U}(-0.2, 0.2)$ or additive shifts $\beta_i^k \leftarrow \beta_i^k + \delta$ where $\delta \sim \mathcal{U}(-10, 10)$ to the piecewise-linear fuel cost coefficients, simulating market price fluctuations; (2) **ramping limits**: we scale both upward and downward ramping constraints $(R_{i,t}^{\uparrow}, R_{i,t}^{\downarrow}) \leftarrow (R_{i,t}^{\uparrow}, R_{i,t}^{\downarrow}) \cdot \gamma$ where $\gamma \sim \mathcal{U}(0.8, 1.2)$, introducing dynamic flexibility or stress in inter-period transitions. After perturbation, we regenerate MPS models using the same data cleaning and model presolve pipeline.

As shown in Figure 2, the runtime for disturbed instances is slightly longer than for normal instances due to increased complexity. Despite this, our framework maintains consistent performance: data cleaning and model presolve achieves a **22× speedup** over the original model, and with learning-driven acceleration, we achieve a **45× overall speedup**. This demonstrates robustness and generalizability under data perturbations and varying problem characteristics.

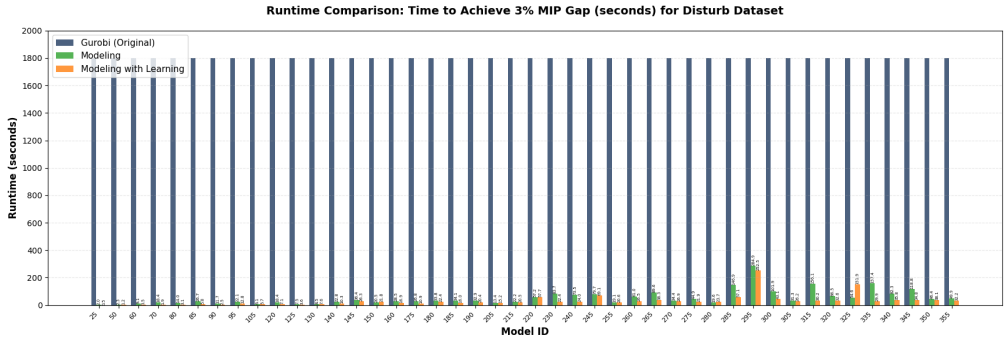


Figure 2: Runtime comparison: Time to achieve 3% MIP gap (seconds) for disturb dataset. The chart compares Gurobi (Original), Modeling Presolve, and Modeling Presovle with Learning (Ours) across half of the disturb instances. Mean runtimes are 1800.00s, 81.55s, and 39.27s respectively. For complete results across all instances, see Table 5 in Section G.

5 Conclusion

We present a recursive self-improvement framework for optimization heuristics where AI systems autonomously refine decision-making through LLM-guided evolution, embodying self-diagnosis, self-critique, and self-update with selective adoption. The framework combines **data cleaning and model presolve** with **recursive self-improvement** to discover interpretable scoring functions, achieving **60× speedup** over commercial solvers while maintaining solution quality. We release an open-source UC/ED benchmark (10–600 units) to support reproducible evaluation. Demonstrated on power system scheduling, the approach generalizes to other combinatorial domains. Future work will explore cross-instance transfer and theoretical guarantees for recursive self-improvement in optimization.

REFERENCES

Erling D Andersen and Knud D Andersen. Presolving in linear programming. *Mathematical programming*, 71(2):221–245, 1995.

Miguel Carrión and José M Arroyo. A computationally efficient mixed-integer linear formulation for the thermal unit commitment problem. *IEEE Transactions on power systems*, 21(3):1371–1378, 2006.

Camille Castera and Peter Ochs. From learning to optimize to learning optimization algorithms. *arXiv preprint arXiv:2405.18222*, 2024.

Abraham Charnes and Carlton E Lemke. Minimization of non-linear separable convex functionals. *Naval Research Logistics Quarterly*, 1(4):301–312, 1954.

Dikshit Chauhan, Bapi Dutta, Indu Bala, Niki van Stein, Thomas Bäck, and Anupam Yadav. Evolutionary computation and large language models: A survey of methods, synergies, and applications. *arXiv preprint arXiv:2505.15741*, 2025.

Tianlong Chen, Weiyi Zhang, Zhou Jingyang, Shiyu Chang, Sijia Liu, Lisa Amini, and Zhangyang Wang. Training stronger baselines for learning to optimize. *Advances in Neural Information Processing Systems*, 33:7332–7343, 2020.

Federal Energy Regulatory Commission et al. Energy primer: A handbook of energy market basics. *Federal Energy Regulatory Commission: Washington, DC, USA*, 2015.

Nonconvex Piecewise Linear Cost. A comparison of mixed-integer programming models for. *Management Science*, 49(9):1268–1273, 2003.

George Dantzig, Selmer Johnson, and Wayne White. A linear programming approach to the chemical equilibrium problem. *Management Science*, 5(1):38–43, 1958.

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 *Proceedings of the aaai conference on artificial intelligence*, volume 34, pp. 1452–1459, 2020.

Antonio Frangioni and Claudio Gentile. Solving nonlinear single-unit commitment problems with ramping constraints. *Operations Research*, 54(4):767–775, 2006.

Len L Garver. Power generation scheduling by integer programming-development of theory. *Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems*, 81(3):730–734, 2008.

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.

Elias B Khalil, Christopher Morris, and Andrea Lodi. Mip-gnn: A data-driven framework for guiding combinatorial solvers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 10219–10227, 2022.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025.

P. Langley. Crafting papers on machine learning. In Pat Langley (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Kaiwen Li, Tao Zhang, Rui Wang, Yuheng Wang, Yi Han, and Ling Wang. Deep reinforcement learning for combinatorial optimization: Covering salesman problems. *IEEE transactions on cybernetics*, 52(12):13142–13155, 2021.

LLC Monitoring Analytics. State of the market report for pjm. *Monitoring Analytics: Norristown, PA, USA*, 2015.

Germán Morales-España, Claudio Gentile, and Andres Ramos. Tight mip formulations of the power-based unit commitment problem. *OR spectrum*, 37(4):929–950, 2015.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Mingjun Pan, Guanquan Lin, You-Wei Luo, Bin Zhu, Zhien Dai, Lijun Sun, and Chun Yuan. Preference optimization for combinatorial optimization problems. *arXiv preprint arXiv:2505.08735*, 2025.

Arsalan Sharifnassab, Saber Salehkaleybar, and Richard Sutton. Metaoptimize: A framework for optimizing step sizes and other meta-parameters. *arXiv preprint arXiv:2402.02342*, 2024.

Dmitry Sorokin and Alexander Kostin. Treedqn: Learning to minimize branch-and-bound tree. *arXiv preprint arXiv:2306.05905*, 2023.

Michael Sucker and Peter Ochs. A generalization result for convergence in learning-to-optimize. *arXiv preprint arXiv:2410.07704*, 2024.

Juan Pablo Vielma, Shabbir Ahmed, and George Nemhauser. Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions. *Operations research*, 58 (2):303–315, 2010.

Allen J Wood, Bruce F Wollenberg, and Gerald B Sheblé. *Power generation, operation, and control*. John wiley & sons, 2013.

Lindong Xie, Genghui Li, Zhenkun Wang, Edward Chung, and Maoguo Gong. Large language model-driven surrogate-assisted evolutionary algorithm for expensive optimization. *arXiv preprint arXiv:2507.02892*, 2025.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. *Advances in neural information processing systems*, 37:43571–43608, 2024.

Tianyu Zhang, Amin Banitalebi-Dehkordi, and Yong Zhang. Deep reinforcement learning for exact combinatorial optimization: Learning to branch. In *2022 26th international conference on pattern recognition (ICPR)*, pp. 3105–3111. IEEE, 2022.

A UC/ED Data and Model Notation

This section provides the complete UC/ED formulation including objective, constraints, and notation.

A.1 Problem Formulation

Objective.

$$\min \sum_t \sum_i \Big(F_{i,t}^{\text{fuel}} + C_i^{\text{SU}}(v_{i,t} + w_{i,t}) + C_i^{\text{on}} u_{i,t} \Big). \tag{1}$$

Constraints

$$\sum_i P_{i,t} = D_t, \quad \forall t, \tag{2}$$

$$u_{i,t}\,\underline{P}_i \le P_{i,t} \le u_{i,t}\,\overline{P}_i, \quad \forall i, t, \tag{3}$$

$$-\,R_{d,i} \le P_{i,t} - P_{i,t-1} \le R_{u,i}, \quad \forall i,\ t > 1, \tag{4}$$

$$\sum_i P_{i,t} \ge D_t, \quad \forall t, \tag{5}$$

$$\sum_i u_{i,t}\,\overline{P}_i \ge D_t, \quad \forall t, \tag{6}$$

$$u_{i,t} - u_{i,t-1} = v_{i,t} - w_{i,t}, \quad \forall i, t, \tag{7}$$

$$P_{i,t} = \sum_{k \in \mathcal{K}_i} \gamma_i^k\, \lambda_{i,t}^k, \forall i, t, \tag{8}$$

$$F_{i,t}^{\text{fuel}} = \sum_{k \in \mathcal{K}_i} \beta_i^k\, \lambda_{i,t}^k, \forall i, t, \tag{9}$$

$$\sum_{k \in \mathcal{K}_i} \lambda_{i,t}^k = u_{i,t}, \forall i, t, \tag{10}$$

$$\sum_{k=0}^{K_i - 1} y_{i,t}^k = u_{i,t}, \forall i, t, \tag{11}$$

$$\lambda_{i,t}^0 \le y_{i,t}^0, \forall i, t, \tag{12}$$

$$\lambda_{i,t}^{K_i} \le y_{i,t}^{K_i - 1}, \forall i, t, \tag{13}$$

$$\lambda_{i,t}^k \le y_{i,t}^{k-1} + y_{i,t}^k, \forall i, t,\ k = 1, \ldots, K_i - 1, \tag{14}$$

$$u_{i,t}, v_{i,t}, w_{i,t}, y_{i,t}^k \in \{0, 1\}, \tag{15}$$

$$P_{i,t}, F_{i,t}^{\text{fuel}}, \lambda_{i,t}^k \ge 0. \tag{16}$$

The formulation captures key operational features: **Load balance** equation 2 matches total generation to demand. **Generation limits** equation 3 bound output based on unit status, while **Ramping limits** equation 4 restrict inter-period output changes. **Power adequacy** equation 5–equation 6 ensures committed capacity covers demand. **Startup/shutdown logic** equation 7 tracks status transitions, and **Piecewise-linear (PWL) fuel cost** constraints equation 8–equation 14 approximate nonlinear cost curves.

A.2 Notation

$u_{i,t}$ on/off status of unit i at time t.

$v_{i,t}, w_{i,t}$ startup/shutdown indicators.

$p_{i,t}$ power output of unit i at time t.

$\underline{P}_i, \overline{P}_i$ minimum / maximum output of unit i.

$R_{u,i}, R_{d,i}$ ramp-up / ramp-down limits of unit i.

D_t system demand at time t.

$F_{i,t}^{\text{fuel}}$ fuel-cost variable.

γ_i^k output level of breakpoint k.

β_i^k fuel cost at breakpoint γ_i^k.

$\lambda_{i,t}^k$ convex-combination weights.

$y_{i,t}^k$ SOS2 adjacency binaries.

$\{0, 1, \ldots, K_i\}$ indices of PWL breakpoints for unit i.

Table 3: Key Physical Parameters and Operational Ranges

Parameter	**Symbol**	**Range / Value**
Max Capacity	$P_{\max}$	$80 \sim 300$ MW
Min Capacity	$P_{\min}$	$15 \sim 60$ MW
Marginal Cost	C_{var}	$15 \sim 30$ \$/MWh
Start-up Cost	C_{start}	$400 \sim 1200$ \$
System Demand	D_t	$600 \sim 1000$ MW

B UC/ED INSTANCE GENERATION

Our benchmark instances are generated using a scalable template-based approach that extends the classic 10-unit UC/ED benchmark to arbitrary problem sizes. The generation process uses 10 distinct unit type templates with realistic technical parameters (minimum/maximum capacity, PWL fuel cost curves, ramp rates) derived from industrial power system data. These templates are cyclically replicated to create instances with I units ranging from 10 to 600. The method ensures consistency with academic benchmarks while maintaining industrial relevance through parameter calibration against FERC and PJM market profiles.

Algorithm 2 UC/ED Instance Generation

Input: number of units I, random seed rng_seed, template patterns
Output: MPS file for UC/ED instance
// Initialize parameters
$J \leftarrow 24$ *(time periods)*
$K \leftarrow 6$ *(PWL segments)*
$UT, DT \leftarrow 4$ *(min up/down time)*
// Template replication
Load 10-unit type patterns: $c_s^{pattern}, l^{pattern}, L^{pattern}, \mu^{pattern}, \lambda^{pattern}, \gamma^{pattern}$
Replicate to I units: $c_s, l, L, \mu, \lambda, \gamma \leftarrow \text{TILEPATTERNS}(patterns, I)$
// Generate random system parameters
$M[j] \leftarrow \text{RANDOMINT}(55, 1717) \quad \forall j \in [1, J]$ *(demand lower bounds)*
$n[j] \leftarrow \text{RANDOMINT}(500, 2800) \quad \forall j \in [1, J]$ *(exact demand)*
$q[i] \leftarrow \text{RANDOMUNIFORM}(-1.0, -0.01) \quad \forall i \in [1, I]$
$q[2] \leftarrow 0.0$ *(special constraint)*
// Generate ramp rates and prices
$ramp\_up[i], ramp\_down[i] \leftarrow 300.0 \quad \forall i$ *(MW/hour)*
$price_t[j] \leftarrow \text{LINSPACE}(200, 600, J)$ *(time-of-use pricing)*
$bigM\_ramp[i] \leftarrow 3000.0 \quad \forall i$
// Optional: Parameter perturbation
if perturbation enabled **then**
  Apply random perturbations to $c_s, l, L, \mu, \lambda, \gamma, ramp\_up, ramp\_down$
  Enforce physical constraints (monotonicity, convexity, $l < L$)
end if
// Build MIP model
Initialize PuLP model with variables: $s_{i,j}$ (binary), $x_{i,j}, p_{i,j}$ (continuous), $u_{i,j,k}, \alpha_{i,j,k}$ (SOS2)
Add constraints:
- Min up/down time: $UT \leq \sum_{k=j}^{j+UT-1} s_{i,k}$ when $startup_{i,j} = 1$
- Output bounds: $l_i \cdot s_{i,j} \leq p_{i,j} \leq L_i \cdot s_{i,j}$
- PWL fuel cost: $p_{i,j} = \sum_k \mu_{i,k} \cdot u_{i,j,k}$, $x_{i,j} = \sum_k \lambda_{i,k} \cdot u_{i,j,k}$
- Demand balance: $\sum_i p_{i,j} = n_j$
- Ramping: $|p_{i,j+1} - p_{i,j}| \leq ramp$ (with Big-M relaxation when unit off)
- Additional constraints from CSV template
// Export
Write model to MPS format
Format numeric precision (8 decimal places, truncation)
return formatted MPS file

Instance Generation Process. The instance generation process in Algorithm 2 follows a systematic workflow to create scalable UC/ED MIP instances:

1. **Template Loading and Replication**: The algorithm loads 10 distinct unit type templates, each containing technical parameters (c_s: startup cost, l: minimum output, L: maximum output, μ: PWL output breakpoints, λ: PWL cost breakpoints, γ: capacity). These templates are cyclically replicated using TILEPATTERNS to create I units, ensuring that unit types are distributed evenly across the instance.
2. **System Parameter Generation**: Random system-level parameters are generated: demand bounds $M[j]$ and exact demand $n[j]$ for each time period $j \in [1, 24]$, unit-specific coefficients $q[i]$, and time-of-use pricing $price_t[j]$ that varies linearly from 200 to 600 across 24 periods.
3. **Unit-Specific Parameters**: Ramp rates ($ramp\_up[i]$, $ramp\_down[i]$) are set to 300 MW/hour for all units, and Big-M values for ramping constraints are set to 3000.0. These values can be optionally perturbed to create variant instances.
4. **Optional Perturbation**: If perturbation is enabled, random multiplicative or additive noise is applied to physical parameters (costs, capacities, ramp rates) while enforcing physical constraints: PWL cost curves must remain monotonic and convex, and minimum output must be less than maximum output ($l_i < L_i$).

5. **MIP Model Construction**: Using PuLP, the algorithm constructs a complete MIP model with binary variables ($s_{i,j}$: unit status, $\alpha_{i,j,k}$: SOS2 adjacency), continuous variables ($p_{i,j}$: power output, $x_{i,j}$: fuel cost, $u_{i,j,k}$: PWL weights), and constraints (min up/down time, output bounds, PWL fuel cost, demand balance, ramping with Big-M relaxation, and additional constraints from CSV templates).
6. **MPS Export and Formatting**: The model is exported to MPS format, and numeric precision is standardized to 8 decimal places using truncation (rounding down) to ensure solver compatibility and reproducibility.

The generation algorithm ensures that each instance maintains realistic power system characteristics: (1) **scalability** through cyclic template replication, allowing generation of instances from 10 to 600 units, (2) **realism** via parameter calibration against industry standards (FERC/PJM profiles), (3) **variability** through randomized demand profiles and optional parameter perturbations, and (4) **computational tractability** by preserving the structural properties (convex PWL costs, linear constraints) that enable efficient MIP solving. The resulting instances provide a comprehensive testbed for evaluating optimization algorithms across different problem scales while maintaining consistency with established benchmarks.

C Automated Presolve Algorithm

This section provides the complete pseudocode and mathematical formulation for the automated presolve process that transforms convex-combination piecewise-linear (PWL) formulations into semi-continuous reformulations. This process is a key component of our pipeline, automatically detecting and rewriting PWL cost blocks to improve sparsity, numerical stability, and solver performance.

Semi-Continuous PWL Reformulation. Direct UC/ED formulations with convex-combination PWL costs are computationally expensive: each generator with K_i cost segments introduces K_i convexity variables and adjacency constraints, inflating problem size and weakening LP relaxations. We replace the standard convex-combination representation with the following semi-continuous multiple-choice formulation:

$$P_{i,t} = \sum_{k=1}^{K_i} w_{i,t}^k, \qquad \forall i, t, \tag{17}$$

$$F_{i,t}^{\text{fuel}} = \sum_{k=1}^{K_i} a^k w_{i,t}^k + \sum_{k=1}^{K_i} b^k \, y_{i,t}^k, \qquad \forall i, t, \tag{18}$$

$$\sum_{k=1}^{K_i} y_{i,t}^k = u_{i,t}, \qquad \forall i, t, \tag{19}$$

$$\gamma_i^{k-1} \, y_{i,t}^k \le w_{i,t}^k \le \gamma_i^k \, y_{i,t}^k, \qquad \forall i, t, k, \tag{20}$$

$$y_{i,t}^k \in \{0, 1\}, \quad w_{i,t}^k \ge 0, \qquad \forall i, t, k. \tag{21}$$

The coefficients are computed as

$$a^k = \frac{\beta_i^k - \beta_i^{k-1}}{\gamma_i^k - \gamma_i^{k-1}}, \qquad b^k = \frac{\beta_i^{k-1}\gamma_i^k - \beta_i^k \gamma_i^{k-1}}{\gamma_i^k - \gamma_i^{k-1}}.$$

This formulation eliminates convexity and adjacency constraints, yields a smaller and better-scaled constraint matrix, and preserves the same feasible (P, F) space and optimal objective as the ideal SOS2 representation (Cost, 2003), while offering superior sparsity and numerical conditioning for large-scale UC/ED instances.

Generalized Tightening of Semi-Continuous Constraints. The formulation in Eq. (1)–(5) provides a general semi-continuous reformulation for PWL cost functions with multiple segments ($K_i > 2$), where each segment is defined by a power interval $[\gamma^{k-1}, \gamma^k]$ and piecewise cost slope

a^k. To improve solver performance, we apply interval reasoning to tighten variable bounds and eliminate redundant segments. Specifically, given a current domain $[L, U]$ of power output $P_{i,t}$, we evaluate each segment's feasibility:

- If $\gamma^{k-1} > L$ and $\gamma^k < U$, then $y^k = 0$.
- If $\gamma^{k-1} \leq L$ and $\gamma^k < U$, then $\gamma^k = L$.
- If $\gamma^{k-1} > L$ and $\gamma^k \geq U$, then $\gamma^{k-1} = U$.
- If $\gamma^{k-1} \leq L$ and $\gamma^k \geq U$, then $\gamma^{k-1} = 0, \gamma^k = \gamma^0$.

These refined bounds yield a sparser model, preserve equivalence to the original PWL formulation, and help strengthen the root relaxation.

Automatic Detection and Rewriting Pipeline. The automated presolve process operates in several stages:

Row classification. We scan the constraint matrix and classify each non-redundant row into one of four types: (i) set-partitioning rows (exactly one binary active), (ii) cumulative rows (unit-continuous columns summing to ± 1), (iii) bound-link rows (one continuous column linked to a binary with ± 1 coefficients), and (iv) convex-combination rows encoding PWL slopes. This exposes the underlying PWL block structure.

Semi-continuous variable discovery. For each bound-link row containing a continuous column v and a binary column y, we compute the tightest bounds (ℓ_0, u_0) under $y = 0$ and (ℓ_1, u_1) under $y = 1$. Infeasible cases directly fix y; otherwise we record

$$\text{semi\_cols\_map}[v][y] = (\ell_0, u_0, \ell_1, u_1).$$

PWL block reconstruction. Starting from each cumulative row, we collect adjacent bound-link rows and their associated binaries, identify a covering set-partitioning row, and reconstruct the ordered sequence of breakpoints by selecting a convex-combination row with monotone induced slopes.

Model rewriting. For each detected PWL block, the original λ-based convex-combination is replaced by semi-continuous variables $w_{i,t}^k$:

$$\gamma_i^{k-1} y_{i,t}^k \leq w_{i,t}^k \leq \gamma_i^k y_{i,t}^k, \qquad P_{i,t} = \sum_{k=1}^{K_i} w_{i,t}^k,$$

and the linear cost is rewritten as

$$F_{i,t}^{\text{fuel}} = \sum_{k=1}^{K_i} a^k w_{i,t}^k + \sum_{k=1}^{K_i} b^k y_{i,t}^k.$$

All convexity, adjacency, and λ-columns are removed.

Bound tightening and scaling. We aggregate bounds across all controlling binaries to derive tighter global bounds and normalize coefficient magnitudes to improve numerical conditioning.

Algorithm 3 Presolve: Convex-Combination PWL → Semi-Continuous Reformulation

Input: constraint matrix M **Output:** presolved matrix $\tilde{M}$
for row r in M **do**
  classify r into set-partitioning, cumulative, convex-combination, or bound-link
  if r is bound-link **then**
    extract (v, y) and compute bounds under $y = 0, 1$
    if infeasible **then**
      fix y
    else
      record $(\ell_0, u_0, \ell_1, u_1)$
    end if
  end if
end for
for each cumulative row **do**
  collect neighbors and reconstruct ordered PWL block
end for
for each detected block **do**
  introduce semi-continuous variables w
  rewrite power balance and fuel cost
  delete convex-combination rows and λ variables
end for
tighten bounds and rescale coefficients
return $\tilde{M}$

The algorithm operates in several stages: (1) row classification to identify structural patterns in the constraint matrix, (2) semi-continuous variable discovery by analyzing bound-link constraints, (3) PWL block reconstruction by connecting related constraints, (4) model rewriting to replace convex-combination formulations with semi-continuous variables, and (5) bound tightening and coefficient scaling to improve numerical conditioning. This automated process eliminates redundant variables and constraints, tightens LP relaxations, and produces compact instances that serve as the entry point for the subsequent LLM+Evolve pipeline.

D Proof of Equivalence between PWL Formulations

In this appendix, we provide a formal proof of the equivalence between the standard convex-combination piecewise-linear (PWL) formulation (referred to as Model CC) and the proposed semi-continuous formulation (referred to as Model SC). The derivation logic follows the transformation principles for mixed-integer programming formulations.

For clarity, we omit the unit and time indices (i, t) and focus on a single generator's cost function. Let $\mathcal{K} = \{1, \ldots, K\}$ denote the set of linear segments. The breakpoints are defined as $\{(\gamma^0, \beta^0), (\gamma^1, \beta^1), \ldots, (\gamma^K, \beta^K)\}$, where γ^k represents power output and β^k represents fuel cost.

Model CC (Convex Combination): Defined by variables $\lambda^k \geq 0$ (weights) and binary variables y^k_{SOS} (adjacency indicators).

$$P = \sum_{k=0}^{K} \gamma^k \lambda^k, \quad F^{\text{fuel}} = \sum_{k=0}^{K} \beta^k \lambda^k, \tag{22a}$$

$$\sum_{k=0}^{K} \lambda^k = 1, \tag{22b}$$

$$\lambda^0 \leq y^0_{SOS}, \quad \lambda^K \leq y^{K-1}_{SOS}, \tag{22c}$$

$$\lambda^k \leq y^{k-1}_{SOS} + y^k_{SOS}, \quad \forall k = 1, \ldots, K-1, \tag{22d}$$

$$\sum_{k=0}^{K-1} y^k_{SOS} = 1, \quad y^k_{SOS} \in \{0, 1\}. \tag{22e}$$

Model SC (Semi-Continuous): Defined by continuous variables $w^k \geq 0$ and binary variables $y^k \in \{0, 1\}$.

$$P = \sum_{k=1}^{K} w^k, \tag{23a}$$

$$F^{\text{fuel}} = \sum_{k=1}^{K} (a^k w^k + b^k y^k), \tag{23b}$$

$$\gamma^{k-1} y^k \leq w^k \leq \gamma^k y^k, \quad \forall k \in \mathcal{K}, \tag{23c}$$

$$\sum_{k=1}^{K} y^k = 1, \quad y^k \in \{0, 1\}. \tag{23d}$$

where $a^k = \frac{\beta^k - \beta^{k-1}}{\gamma^k - \gamma^{k-1}}$ and $b^k = \frac{\beta^{k-1}\gamma^k - \beta^k \gamma^{k-1}}{\gamma^k - \gamma^{k-1}}$.

Model CC and Model SC are equivalent: any feasible solution (P, F^{fuel}) in one formulation maps to a feasible solution in the other with the same objective value.

Direction 1: Model SC ⇒ Model CC

Let $(\mathbf{w}, \mathbf{y})$ be a feasible solution to Model SC. From equation 23d, exactly one binary variable is active. Let $y^m = 1$ for a specific segment $m \in \{1, \ldots, K\}$, and $y^k = 0$ for all $k \neq m$. Constraint equation 23c implies $w^k = 0$ for $k \neq m$, and $\gamma^{m-1} \leq w^m \leq \gamma^m$. The power output is $P = w^m$.

We construct a solution for Model CC as follows: Define $\alpha = \frac{w^m - \gamma^{m-1}}{\gamma^m - \gamma^{m-1}}$. Since $\gamma^{m-1} \leq w^m \leq \gamma^m$, we have $0 \leq \alpha \leq 1$. Set the weights λ as:

$$\lambda^m = \alpha, \quad \lambda^{m-1} = 1 - \alpha, \quad \lambda^k = 0 \ (\forall k \notin \{m-1, m\}). \tag{24}$$

Set the SOS2 binaries as $y_{SOS}^{m-1} = 1$ and all others to 0.

1. **Feasibility Check:**
   - $\sum \lambda^k = (1 - \alpha) + \alpha = 1$. (Satisfies equation 22b).
   - The non-zero λ indices are $\{m - 1, m\}$, corresponding to the active SOS2 interval $m - 1$. (Satisfies adjacency constraints).
   - Power $P_{CC} = \lambda^{m-1}\gamma^{m-1} + \lambda^m \gamma^m = (1 - \alpha)\gamma^{m-1} + \alpha\gamma^m$. Substituting α, this simplifies to $P_{CC} = w^m$, matching Model SC.
2. **Cost Check:** The cost in Model SC is $F_{SC} = a^m w^m + b^m$. Substituting $w^m = (1 - \alpha)\gamma^{m-1} + \alpha\gamma^m$:

   $$\begin{aligned} F_{SC} &= a^m[(1 - \alpha)\gamma^{m-1} + \alpha\gamma^m] + b^m(1 - \alpha + \alpha) \\ &= (1 - \alpha)(a^m\gamma^{m-1} + b^m) + \alpha(a^m\gamma^m + b^m). \end{aligned}$$

   Recall that $y = a^m x + b^m$ is the line equation passing through $(\gamma^{m-1}, \beta^{m-1})$ and (γ^m, β^m). Therefore:

   $$a^m\gamma^{m-1} + b^m = \beta^{m-1} \quad \text{and} \quad a^m\gamma^m + b^m = \beta^m.$$

   Thus, $F_{SC} = (1 - \alpha)\beta^{m-1} + \alpha\beta^m$, which is exactly $\sum \beta^k \lambda^k = F_{CC}$.

Direction 2: Model CC ⇒ Model SC

Let $(\boldsymbol{\lambda}, \mathbf{y}_{SOS})$ be a feasible solution to Model CC. Due to the SOS2 constraints, non-zero λ values can only exist at indices $\{m - 1, m\}$ corresponding to an active $y_{SOS}^{m-1} = 1$. Thus, $\lambda^{m-1} + \lambda^m = 1$, and $P = \lambda^{m-1}\gamma^{m-1} + \lambda^m\gamma^m$.

We construct a solution for Model SC as follows: Set $y^m = 1$ (select segment m) and all other $y^k = 0$. Set $w^m = P$ and all other $w^k = 0$.

1. **Feasibility Check:** Since P is a convex combination of γ^{m-1} and γ^m, it holds that $\gamma^{m-1} \leq w^m \leq \gamma^m$. This satisfies the semi-continuous bounds equation 23c given $y^m = 1$.

2. **Cost Check:** Using the linearity derived in Direction 1, the cost of any point $P \in [\gamma^{m-1}, \gamma^m]$ on the line segment defined by parameters a^m, b^m is given by $a^m P + b^m$. Since Model CC defines cost as the convex combination of endpoints β^{m-1}, β^m, and the line segment connects these endpoints, the values are identical.

E MAIN PIPELINE: SCORING AND WARM START

This section details the main pipeline that uses evolved configurations to score variables, construct subproblems, and warm-start the master problem. All parameters (scoring function parameters, RC threshold, and fixing rate) are obtained from the evolved configuration θ through the recursive self-improvement process described in Algorithm 5.

Variable Scoring and Fixing. The pipeline constructs subproblems using the evolved configuration θ obtained from Algorithm 5, which includes both scoring function parameters (α_1) and hyperparameters ($\hat{\theta}_{\text{RC}}, \hat{\theta}_{\text{FR}}$). Both the scoring function and hyperparameters (RC threshold and fixing rate) are evolved together through the LLM-guided evolution process. After solving the LP relaxation, the evolved scoring function ranks all candidate variables, and the top-k variables ($k = \lfloor \hat{\theta}_{\text{FR}} \times |\mathcal{C}| \rfloor$) are fixed based on the evolved RC threshold ($RC > \hat{\theta}_{\text{RC}}$ fixes to 0, $RC < -10$ fixes to 1, $|RC| < 10^{-6}$ fixes based on LP value). After constraint propagation, the restricted subproblem is solved to find a feasible solution for warm-starting the master problem.

Algorithm 4 Scoring and Subproblem Solving with Evolved Configuration

Input: presolved model $\tilde{\mathcal{M}}$, evolved configuration θ (including scoring parameters α_1 and hyperparameters $\hat{\theta}_{\text{RC}}, \hat{\theta}_{\text{FR}}$), time limits $T_{\text{sub}}, T_{\text{master}}$
Output: warm-started solution Sol_0
// LP relaxation and scoring
$(x^{LP}, rc) \leftarrow \text{SOLVELP}(\tilde{\mathcal{M}})$
$\mathcal{C} \leftarrow$ collect binary variables with (x^{LP}, rc)
for each $y_{u,t} \in \mathcal{C}$ **do**
  score$(y_{u,t}) \leftarrow \alpha_1 \cdot \log(|rc_{u,t}| + 1)$
end for
Sort $\mathcal{C}$ by score descending; $k \leftarrow \lfloor \hat{\theta}_{\text{FR}} \times |\mathcal{C}| \rfloor$
$\mathcal{C}_{\text{top}} \leftarrow$ top-k variables from sorted $\mathcal{C}$
// Variable fixing and subproblem solving
$\mathcal{F} \leftarrow \emptyset$
for each $y_{u,t} \in \mathcal{C}_{\text{top}}$ **do**
  if $rc_{u,t} > \hat{\theta}_{\text{RC}}$ **then**
    fix $y_{u,t} = 0$, add to $\mathcal{F}$
  else if $rc_{u,t} < -10$ **then**
    fix $y_{u,t} = 1$, add to $\mathcal{F}$
  else if $|rc_{u,t}| < 10^{-6}$ **then**
    fix based on $x^{LP}_{u,t}$, add to $\mathcal{F}$
  end if
end for
Apply constraint propagation
SubSol $\leftarrow \text{SOLVESUBPROBLEM}(\tilde{\mathcal{M}}, \mathcal{F}, T_{\text{sub}})$
// Warm start master problem
$\text{Sol}_0 \leftarrow \text{WARMSTARTMASTER}(\tilde{\mathcal{M}}, \text{SubSol}, T_{\text{master}})$
return Sol_0

Warm Start of the Master Problem. The feasible solution from the subproblem is injected into the master problem via Gurobi's `Start` attribute, providing an initial incumbent and upper bound that enables aggressive pruning and accelerates convergence. The evolved configuration θ determines both the scoring function (used to rank variables) and the hyperparameters (RC threshold $\hat{\theta}_{\text{RC}}$

and fixing rate $\hat{\theta}_{\text{FR}}$) used to select and fix variables, demonstrating that all components are learned together through the evolutionary process.

F EVOLVE PSEUDOCODE (HEURPY)

Algorithm 5 Evolve: Lightweight Evolutionary Refinement of LLM-Proposed Scoring Functions (from `heur.py`)

Input: presolved MIP $\tilde{\mathcal{M}}$, LP relaxation model $\tilde{\mathcal{M}}^{LP}$, evolution budget $(pop, gens, \mu)$, time limits $(T_{\text{sub}}, T_{\text{master}})$
Output: best configuration $cfg^\star$ and its warm-started master solution
// Candidate features from LP
$(x^{LP}, rc) \leftarrow$ SOLVELP($\tilde{\mathcal{M}}^{LP}$)
$\mathcal{C} \leftarrow$ collect binary variables with features $(name, lpv, rc, near01, \text{period\_bonus}, \text{conflict\_risk})$ *(near01 $= 0.5 - \min(lpv, 1 - lpv)$, period_bonus, conflict_risk)*
// Initialize population $\mathcal{P}$ with multi-feature scoring functions
$\mathcal{P} \leftarrow \emptyset$
if LLM API key available **then**
  $cfg_0 \leftarrow$ QUERYLLMFORSEEDSCORER() *(initial config with $\alpha_1, \alpha_2, \alpha_3, \alpha_4$)*
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{cfg_0\}$
end if
$\mathcal{P} \leftarrow \mathcal{P} \cup$ NONLINEARSEEDPOOL() *(diverse nonlinear scoring functions)*
fill remaining slots in $\mathcal{P}$ by Gaussian perturbation *(sample $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ and fix_rate using Gaussian noise: $gauss(0.0, \sigma)$)*
// Evolution loop
for $g = 1$ to $gens$ **do**
  $\mathcal{S} \leftarrow \emptyset$
  for each $cfg \in \mathcal{P}$ **do**
    $scorer \leftarrow$ COMPILESCORER(cfg.scoring_code) *(compile Python scoring function)*
    $\mathcal{F} \leftarrow$ SELECTFIXSET($\mathcal{C}, cfg, scorer$)
    SubSol $\leftarrow$ SOLVESUBPROBLEM($\tilde{\mathcal{M}}, \mathcal{F}, T_{\text{sub}}$)
    MasterSol $\leftarrow$ WARMSTARTMASTER($\tilde{\mathcal{M}}$, SubSol, T_{master})
    $fit \leftarrow$ FITNESS(MasterSol) *(hierarchical: gap-based penalty or efficiency score)*
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{(cfg, fit)\}$
  end for
  Parents $\leftarrow$ TOPMU($\mathcal{S}, \mu$)
  $\mathcal{P} \leftarrow$ Parents
  // Refill by LLM-guided mutation based on fitness feedback
  while $|\mathcal{P}| < pop$ **do**
    pick $p \in$ Parents uniformly
    $child \leftarrow$ LLM-MUTATE(p, performance_feedback) *(LLM receives parent config and fitness metrics, can update $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ weights or generate new `scoring_code`)*
    (optionally retry to enforce valid nonlinear `scoring_code` if generated)
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{child\}$
  end while
end for
$cfg^\star \leftarrow \arg\max_{(cfg, fit) \in \mathcal{S}} fit$
return $cfg^\star$, MasterSol($cfg^\star$)

Fitness Function Design. The fitness function in Algorithm 5 employs a *hierarchical evaluation strategy* that prioritizes solution quality (optimality gap) over raw runtime efficiency. This design reflects the practical requirement that UC/ED optimization must achieve acceptable gap thresholds before optimizing for speed. Specifically, the fitness is computed as follows:

1. **Gap-satisfied regime** (when the master problem's final MIP gap ≤ 0.001):
   - The fitness uses an *efficiency score* measuring computational cost:

$$\text{score} = \log(1 + \text{runtime}) + 0.01 \cdot \log(1 + \text{node\_count})$$

   where smaller scores indicate faster convergence. This score is then negated to convert it into a fitness value (larger is better for evolution).

- The logarithmic transformation ensures that the fitness remains stable across orders of magnitude differences in runtime and node counts, while the small weight on node count (0.01) keeps the focus primarily on wall-clock time.

2. **Gap-unsatisfied regime** (when the final MIP gap > 0.001):
   - The fitness uses a *penalty score* that strongly discourages configurations failing to reach the gap threshold:

$$\text{penalty} = 1000 + 40 \cdot \text{rel\_init\_quality} + 60 \cdot \text{gap}$$

   where

$$\text{rel\_init\_quality} = \begin{cases} \dfrac{|\text{init\_solution} - \text{dual\_bound}|}{|\text{dual\_bound}| + 1} & \text{if dual\_bound is valid} \\ \text{gap} & \text{otherwise} \end{cases}$$

   This penalty is also negated to convert it into a fitness value (larger is better).
   - The base offset (1000) ensures that all gap-unsatisfied configurations receive a fitness lower than any gap-satisfied configuration (since typical efficiency scores are much smaller than 1000). The relative weights (40 on initial solution quality, 60 on gap) emphasize final gap as the primary concern, while the initial solution quality from the subproblem serves as a secondary indicator of scoring function effectiveness.

This hierarchical design ensures that the evolutionary process first converges to configurations that reliably achieve the target gap threshold, and then optimizes for computational efficiency among those satisfactory configurations. The objective value (ObjVal) is *not* directly used in the fitness computation; instead, it influences the fitness only indirectly through the relative initial solution quality term in the penalty regime, which measures how well the warm-start solution compares to the LP dual bound.

LLM-Guided Mutation and Prompt Design. The mutation operator in Algorithm 5 uses *LLM-guided mutation* rather than traditional Gaussian perturbation. Specifically, during population initialization, Gaussian noise ($gauss(0.0, \sigma)$) is applied to sample random configurations for $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ and fix_rate. However, during the evolution loop, new candidates are generated by calling LLM-MUTATE, which queries an LLM API with a carefully designed prompt containing:

- **Parent configuration**: The selected parent's scoring function parameters or code
- **Performance feedback**: Detailed metrics including best objective found, improvement, feasibility status, iteration count, and recent performance trends
- **Stage-specific constraints**: Based on iteration count, the prompt enforces either (1) linear stage (iteration < 800): fixed form $\alpha_1 \cdot \log(|RC| + 1)$, only α_1 can be evolved, or (2) nonlinear stage (iteration ≥ 800): LLM can generate new `scoring_code` following predefined templates
- **Evolution guidance**: Instructions to minimize subproblem objective, prioritize feasibility if infeasible, and adjust fix_rate based on feasibility feedback

The LLM receives this comprehensive context and generates a new configuration that can either (1) update weight parameters ($\alpha_1, \alpha_2, \alpha_3, \alpha_4$), or (2) generate a completely new `scoring_code` with different structural combinations. This LLM-guided approach enables intelligent, context-aware mutations that adapt based on performance feedback, rather than random Gaussian perturbations.

G MIP TIME COMPARISON

This section provides complete runtime comparison results across all instances. Table 4 shows results for normal instances, and Table 5 shows results for disturbed instances.

Table 4: Runtime comparison: Time to achieve 3% MIP gap (seconds) for normal dataset

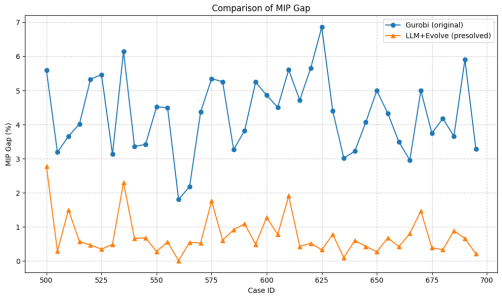
Gurobi vs Our Method (Part 1)				Gurobi vs Our Method (Part 2)			
Model ID	**Gurobi (s)**	**Gurobi Pre (s)**	**Ours (s)**	**Model ID**	**Gurobi (s)**	**Gurobi Pre (s)**	**Ours (s)**
64	1800.00	5.31	2.25	295	1800.00	82.23	78.52
67	1800.00	4.00	1.33	298	1800.00	80.20	30.90
70	1800.00	8.49	2.96	301	1800.00	25.60	24.36
73	1800.00	2.39	1.77	300	1800.00	18.22	26.02
79	1800.00	11.08	7.16	305	1800.00	39.24	35.04
82	1800.00	11.11	4.03	310	1800.00	21.08	22.27
100	1800.00	17.44	5.76	315	1800.00	41.83	25.06
103	1800.00	14.77	4.81	320	1800.00	89.22	67.95
106	1800.00	22.74	8.25	325	1800.00	67.27	34.87
109	1800.00	4.39	4.66	330	1800.00	72.31	53.09
121	1800.00	6.78	5.89	335	1800.00	33.36	23.01
124	1800.00	21.69	9.39	340	1800.00	73.59	36.55
130	1800.00	30.87	12.44	345	1800.00	249.65	240.07
133	1800.00	5.48	3.33	350	1800.00	39.35	35.22
139	1800.00	29.05	22.79	355	1800.00	76.12	42.83
145	1800.00	6.10	6.66	360	1800.00	98.36	30.19
148	1800.00	19.45	13.50	365	1800.00	51.00	35.62
151	1800.00	19.32	13.36	370	1800.00	37.81	37.19
154	1800.00	37.56	12.37	375	1800.00	68.68	33.44
160	1800.00	13.26	8.24	380	1800.00	75.79	21.79
163	1800.00	19.38	16.80	385	1800.00	98.05	64.78
166	1800.00	29.31	13.42	390	1800.00	89.92	30.63
175	1800.00	23.43	11.82	395	1800.00	110.63	15.94
178	1800.00	17.23	10.73	400	1800.00	24.28	12.75
181	1800.00	32.57	12.49	405	1800.00	20.15	12.91
184	1800.00	43.12	22.05	410	1800.00	142.99	99.91
187	1800.00	52.65	17.06	415	1800.00	38.88	27.05
190	1800.00	25.01	24.84	420	1800.00	45.44	33.43
196	1800.00	17.11	11.27	425	1800.00	88.41	16.40
199	1800.00	43.74	35.47	430	1800.00	16.88	15.52
202	1800.00	20.28	23.60	435	1800.00	111.66	129.92
205	1800.00	14.84	12.95	440	1800.00	1800.26	19.14
208	1800.00	22.21	17.45	445	1800.00	29.88	17.66
211	1800.00	36.13	24.67	450	1800.00	30.28	40.63
214	1800.00	55.63	17.41	460	1800.00	54.80	26.72
220	1800.00	16.86	17.99	465	1800.00	69.81	107.28
223	1800.00	64.26	54.14	470	1800.00	77.61	23.71
226	1800.00	27.07	22.10	475	1800.00	28.71	21.02
229	1800.00	57.08	18.47	480	1800.00	38.74	47.56
232	1800.00	19.41	14.49	485	1800.00	15.66	22.58
235	1800.00	95.38	53.98	490	1800.00	32.59	32.16
238	1800.00	41.29	21.81	495	1800.00	156.98	51.93
241	1800.00	34.84	25.14	500	1800.00	16.52	17.38
244	1800.00	17.43	19.21	505	1800.00	49.14	49.67
247	1800.00	30.16	27.66	510	1800.00	30.36	31.44
250	1800.00	47.66	54.58	515	1800.00	103.42	25.88
253	1800.00	28.18	24.96	520	1800.00	16.22	21.40
259	1800.00	16.95	17.55	525	1800.00	247.89	68.83
262	1800.00	88.14	23.45	530	1800.00	20.03	21.95
265	1800.00	25.93	21.90	535	1800.00	167.73	31.50
271	1800.00	16.65	20.51	540	1800.00	47.50	25.98
274	1800.00	46.65	23.64	545	1800.00	1090.86	22.61
277	1800.00	37.62	35.17	550	1800.00	84.05	43.27
280	1800.00	26.57	30.27	555	1800.00	17.72	32.36
283	1800.00	79.56	51.54	560	1800.00	169.53	28.92
286	1800.00	71.74	29.89	565	1800.00	44.68	22.41
289	1800.00	42.95	55.67	570	1800.00	46.19	24.33
292	1800.00	90.91	57.15	575	1800.00	189.67	104.75
				580	1800.00	54.58	29.48
				585	1800.00	117.11	42.16
				590	1800.00	120.91	29.51
				595	1800.00	150.18	94.06
				Mean	**1800**	**75.13**	**30.96**

Table 5: Runtime comparison: Time to achieve 3% MIP gap (seconds) for disturb dataset

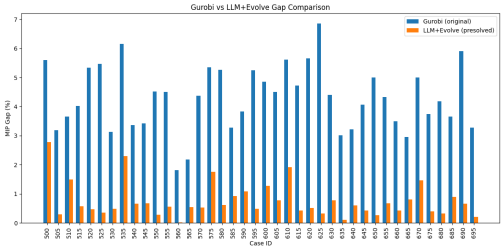
Part 1				Part 2			
Model ID	**Gurobi (s)**	**Gurobi Pre (s)**	**Ours (s)**	**Model ID**	**Gurobi (s)**	**Gurobi Pre (s)**	**Ours (s)**
25	1800.00	2.03	0.49	365	1800.00	152.20	46.54
50	1800.00	2.32	1.18	370	1800.00	59.50	42.81
60	1800.00	9.07	3.47	375	1800.00	42.44	41.06
70	1800.00	18.37	1.95	380	1800.00	39.17	48.62
80	1800.00	14.02	3.14	385	1800.00	40.38	14.37
85	1800.00	26.73	5.83	395	1800.00	121.00	66.32
90	1800.00	11.65	2.54	400	1800.00	38.60	27.23
95	1800.00	22.07	12.80	405	1800.00	96.69	98.35
105	1800.00	6.08	5.71	410	1800.00	63.69	18.77
120	1800.00	18.38	7.12	415	1800.00	140.55	155.89
125	1800.00	7.54	3.60	425	1800.00	25.57	25.12
130	1800.00	8.51	7.57	430	1800.00	40.38	23.07
140	1800.00	22.76	10.26	435	1800.00	265.63	156.15
145	1800.00	36.38	26.26	440	1800.00	27.68	21.18
150	1800.00	20.54	21.82	445	1800.00	93.61	111.67
160	1800.00	28.46	16.86	450	1800.00	121.95	127.68
175	1800.00	26.77	10.93	455	1800.00	118.76	28.57
180	1800.00	33.78	22.44	460	1800.00	97.56	108.65
185	1800.00	34.08	16.03	465	1800.00	81.58	37.83
190	1800.00	32.87	20.39	470	1800.00	85.98	59.39
205	1800.00	13.40	15.25	475	1800.00	29.04	21.94
215	1800.00	22.24	20.53	480	1800.00	43.52	20.57
220	1800.00	57.21	57.72	485	1800.00	35.41	26.54
230	1800.00	83.68	22.37	490	1800.00	118.15	28.92
240	1800.00	73.47	23.96	500	1800.00	431.87	25.78
245	1800.00	79.69	69.07	505	1800.00	122.97	20.47
255	1800.00	19.07	20.55	510	1800.00	58.15	25.92
260	1800.00	60.96	26.49	515	1800.00	46.50	19.37
265	1800.00	89.62	38.34	520	1800.00	70.75	42.56
270	1800.00	34.43	26.95	525	1800.00	30.00	18.82
275	1800.00	44.91	21.86	530	1800.00	315.48	59.49
280	1800.00	19.64	23.71	535	1800.00	43.99	38.86
285	1800.00	146.90	57.14	540	1800.00	157.83	25.52
295	1800.00	284.85	252.54	545	1800.00	70.91	25.70
300	1800.00	101.93	43.13	550	1800.00	54.13	29.48
305	1800.00	31.33	26.17	555	1800.00	36.99	23.75
315	1800.00	156.10	30.25	560	1800.00	266.33	115.11
320	1800.00	66.54	32.57	565	1800.00	71.39	32.39
325	1800.00	54.61	151.94	570	1800.00	202.91	73.71
335	1800.00	157.38	29.85	575	1800.00	60.20	31.23
340	1800.00	82.29	35.79	580	1800.00	173.76	61.33
345	1800.00	118.77	34.82	585	1800.00	154.98	31.91
350	1800.00	46.43	38.10	590	1800.00	58.32	31.63
355	1800.00	49.92	32.20	595	1800.00	530.53	32.49
360	1800.00	42.93	40.14				
				Mean	**1800**	**81.55**	**39.27**

H MIP GAP COMPARISON

This section provides complete MIP gap comparison results across all instances. Figure 3 shows visual comparisons, and Table 6 provides detailed gap values for all instances.



(a) Line plot: Comparison of MIP Gap across cases.



(b) Bar chart: Gurobi vs. LLM+Evolve MIP Gap.

Figure 3: Comparison of MIP gap (%) between **Gurobi (original)** and **Modeling with Learning** across benchmark MPS cases.
(a) Line plot highlights the per-case gap trend.
(b) Bar chart emphasizes the relative reduction of gap by LLM+Evolve.

Table 6: MIP gap (%) comparison across all models. Each row shows the model ID, Gurobi's gap on the original problem, and the gap after applying our LLM+Evolve presolve pipeline.

MODEL ID	GUROBI (ORIGINAL)	LLM+EVOLVE (PRESOLVED)
500	5.60	2.78
505	3.19	0.29
510*	3.66	1.50
515	4.02	0.57
520	5.33	0.47
525	5.47	0.35
530	3.13	0.49
535	6.15	2.30
540	3.36	0.66
545*	3.42	0.68
550	4.52	0.28
555	4.50	0.56
560	1.81	0.01
565	2.18	0.55
570	4.37	0.53
575	5.35	1.76
580	5.26	0.61
585	3.27	0.92
590	3.83	1.09
595	5.25	0.49
600	4.86	1.28
605*	4.51	0.78
610	5.61	1.91
615	4.72	0.43
620	5.66	0.52
625	6.86	0.33
630	4.40	0.78
635	3.02	0.10
640	3.22	0.60
645	4.07	0.43
650	5.00	0.27
655	4.33	0.68
660	3.50	0.42
665	2.96	0.81
670*	5.00	1.47
675	3.75	0.39
680	4.18	0.33
685	3.66	0.89
690	5.91	0.66
695	3.28	0.21