EvoOptiGraph: Weakness-Driven Coevolution via Graph-Based Structural Generation for Optimization Modeling

Qingcan Kang[1*], **Mingyang LIU**[2*], **Xiaojin Fu**[1], **Shixiong Kai**[1], **Tao Zhong**[1], **Mingxuan Yuan**[1†]
[1]Huawei Noah's Ark Lab, Hong Kong
[2]Department of Computer Science, City University of Hong Kong
{kangqingcan,fuxiaojin,kaishixiong,zhongtao5,Yuan.Mingxuan}@huawei.com
mingyaliu8-c@my.cityu.edu.hk

Abstract

Automating optimization modeling from natural language with large language models (LLMs) faces two key challenges. First, training corpora lack structural diversity. Second, data generation pipelines remain static and decoupled from model learning. To address these challenges, we propose **EvoOptiGraph**, a novel framework where data and model co-evolve, driven by model weaknesses. EvoOptiGraph represents each mixed-integer linear program (MILP) as an attributed bipartite graph and applies validity-preserving evolutionary operators to generate structurally diverse instances. The evolved graphs are converted into solver code and natural language via deterministic compilation and verified back-translation. Training proceeds in two stages: supervised fine-tuning (SFT) on an initial dataset, followed by reinforcement learning with verifiable rewards (RLVR), where graph-derived weakness signals guide the generation of new instances targeting the model's failures. This forms a closed loop that continuously updates the training distribution. Empirical results on six public datasets show that EvoOptiGraph significantly outperforms larger generalist models, agentic methods, and specialized baselines in accuracy, executability, and generalization. These results demonstrate that targeted data–model coevolution is an effective strategy for improving LLMs on optimization modeling tasks.

1 Introduction

Optimization modeling converts a natural-language decision problem into a formal mathematical program that can be solved by an optimization engine. This capability is central to applications in logistics, manufacturing, energy, and finance [Singh, 2012, Antoniou and Lu, 2007], but in practice it still depends heavily on human expertise. A modeler must identify decision variables, objectives, constraints, and domain assumptions, then translate them into solver-executable code. Automating this process is therefore an appealing but demanding testbed for large language models (LLMs). Recent work has explored prompt engineering [Ramamonjison et al., 2023], multi-agent reasoning [AhmadiTeshnizi et al., 2024, Xiao et al., 2023], and domain-specific fine-tuning [Huang et al., 2025, Jiang et al., 2024] to improve natural-language-to-optimization modeling.

Two challenges limit current progress. First, training data for optimization modeling is narrow relative to the hypothesis space of valid formulations. Benchmarks such as NL4Opt [Ramamonjison et al., 2023] and MAMO [Huang et al., 2024] are valuable, but they cover only a small subset of the

[*] Equal contribution.
† Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

structural patterns encountered in mathematical programming [Xiao et al., 2025]. Existing synthesis pipelines mainly vary coefficients or textual phrasing within fixed templates, which improves scale but not structural breadth. Text-only evolution methods can generate lexical diversity, yet they offer weak control over the underlying optimization structure [Li et al., 2024]. Second, most training pipelines are static: a dataset is constructed once, and the model is then optimized against that frozen distribution. Such a setup makes it difficult to identify which structural regimes the model fails on and to generate new instances that specifically target those weaknesses.

We address these limitations with **EvoOptiGraph**, a weakness-driven coevolution framework of data and models that couples data generation and model training through an explicit structural representation. Our starting point is to represent each mixed-integer linear program (MILP) as an attributed bipartite graph whose nodes correspond to variables and constraints, and whose edges encode nonzero coefficients. This representation is expressive enough to recover solver-executable code, but structured enough to support validity-preserving genetic operations and fine-grained failure analysis. EvoOptiGraph uses this graph space in two ways: it evolves new optimization instances beyond parameter-level perturbations, and it diagnoses model weaknesses through graph-derived structural features computed on held-out validation problems. The training loop is closed rather than static. We first warm-start the model with supervised fine-tuning (SFT) on an initial population of verified instances. We then estimate a weakness profile for the current model by relating graph features to observed failures. That weakness profile guides further graph evolution, generating instances that are diverse, solvable, and structurally aligned with the model's current blind spots. The newly evolved instances then feed into a reinforcement learning stage with verifiable rewards (RLVR), where rewards are defined by solver executability and solution correctness. In this way, the training distribution adapts as the model changes.

Our contributions are threefold:

- **We propose a novel closed-loop coevolution framework for optimization modeling.** Instead of separating data synthesis from model optimization, EvoOptiGraph links them through graph-level weakness analysis, enabling the training distribution to adapt to the model's evolving failure modes.
- **We design a novel controllable graph-based generator for optimization instances.** Crossover and mutation operate directly on attributed bipartite graphs, enabling structural-level evolution beyond parameter perturbations. Each evolved graph is deterministically compiled into solver code and back-translated into natural language, supporting systematic exploration of diverse optimization structures.
- **We develop an effective weakness-driven model adaptation with verifiable feedback.** After supervised fine-tuning, we use graph-derived weakness signals to evolve new instances that target the model's failures, and optimize the model with RLVR, where rewards are defined by solver executability and solution correctness.

Experiments on six public benchmarks, including ComplexOR, MAMO, and NL4Opt, show that our framework improves optimization-modeling performance over strong baselines and yields better generalization to structurally challenging instances. Ablation studies further indicate that both components of the loop matter: graph-based structural evolution expands the support of the training data, while weakness-driven generation makes that expansion relevant to the model's actual errors.

2 Related Work

2.1 LLMs for Optimization Modeling

Recent work studies how LLMs translate natural-language problem statements into mathematical formulations or solver-executable code. Early efforts relied on prompt-based pipelines, such as the NL4Opt benchmark [Ramamonjison et al., 2023], which highlighted both the promise of LLMs and their tendency to omit constraints or hallucinate assumptions. Subsequent work improves inference-time reasoning via decomposition, multi-agent collaboration, and expert-style prompting, including OptiMUS [AhmadiTeshnizi et al., 2024], Chain-of-Experts [Xiao et al., 2023], and AutoFormulation [Astorga et al., 2024]. In parallel, task-specific models are developed through supervised fine-tuning, such as ORLM [Huang et al., 2025] and LLMOPT [Jiang et al., 2024].

These approaches improve prompting, reasoning, or task adaptation under a fixed training distribution. In contrast, we focus on constructing and iteratively adapting the data distribution to expose structurally informative instances during training. EvoOptiGraph is complementary to prior modeling architectures and can be combined with stronger inference-time reasoning.

2.2 Data Synthesis for Optimization Modeling

Data quality is a key bottleneck for optimization-modeling LLMs [Xiao et al., 2025]. A common approach is parametric template expansion, where canonical problem families are instantiated by varying sizes, coefficients, or templates [Lu et al., 2025]. While controllable and verifiable, such methods generate instances close to fixed structural templates. Recent work explores broader synthesis: MILP-Evolve [Li et al., 2024] iteratively evolves MILP classes via text-based operators on seed problems, OptMATH [Lu et al., 2025] constructs aligned triplets via bidirectional generation and solution matching, and MIPLIB-NL [Li et al., 2026] reverse-generates natural-language descriptions for real-world MILP instances. Other works focus on generating MILP instances primarily for solver evaluation and algorithm stress-testing (e.g., G2MILP [Geng et al., 2023, Alipour et al., 2023], MILP-StuDio [Liu et al., 2024d], and instance-space evolution [Smith-Miles and Bowly, 2015, Liu et al., 2024b, Bowly, 2019]), while our goal is to create training data for LLM-based modeling from natural language.

Our work is closely related to structure-aware instance generation but differs in two aspects. First, EvoOptiGraph applies genetic operations directly on graph representations that preserve optimization structure, rather than relying on textual mutation or parameter perturbation. Second, generation is adaptive: instances are scored by alignment with model weaknesses, enabling dynamic rather than static synthesis.

2.3 Model Training and Reinforcement Learning

SFT is the standard starting point, but token-level likelihood does not directly optimize solver correctness or executability. RL with external verification is therefore widely adopted, including outcome-based rewards [Chen et al., 2025], process supervision with reward models [Dai et al., 2024, Zhu et al., 2025], and preference-based objectives such as DPO [Zheng et al., 2025a]. Other approaches combine LLMs with search strategies, including beam search, Monte Carlo Tree Search, and evolutionary self-improvement [Wang et al., 2025, Zheng et al., 2025b, Liu et al., 2024c, Ye et al., 2024]. For operations research tasks, OR-PRM evaluates intermediate reasoning steps [Wang et al.]. However, these methods typically rely on fixed training datasets and do not adapt data distributions to model weaknesses.

EvoOptiGraph builds on this line of work but shifts the focus to data generation. We also optimize with verifiable rewards, but instead of sampling from a static pool, we diagnose structural error patterns and evolve new instances that target them, turning RL into an adaptive data-model coevolution loop.

3 Methodology

3.1 Problem Formulation and Graph Representation

We study *natural-language-to-optimization modeling*: given a problem description $q \in \mathcal{Q}$, where $\mathcal{Q}$ denotes the space of natural-language descriptions, the goal is to generate a correct optimization model together with its solver-executable implementation. We focus on mixed-integer linear programs (MILPs), which cover a large family of practical decision problems. A MILP can be written as

$$\begin{aligned} \min / \max \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \bowtie \mathbf{b}, \\ & \mathbf{l} \le \mathbf{x} \le \mathbf{u}, \\ & x_i \in \mathbb{Z} \text{ for } i \in \mathcal{I}, \quad x_j \in \mathbb{R} \text{ for } j \in \mathcal{R}, \end{aligned}$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ is the decision vector, $\mathbf{c}$ is the objective coefficient vector, $\mathbf{A}$ is the constraint matrix, $\mathbf{b}$ is the right-hand-side vector, and $\bowtie \in \{\le, =, \ge\}^m$ specifies the constraint senses. Variable bounds are denoted by $\mathbf{l}$ and $\mathbf{u}$, and $\mathcal{I}$ and $\mathcal{R}$ index integer and continuous variables, respectively.

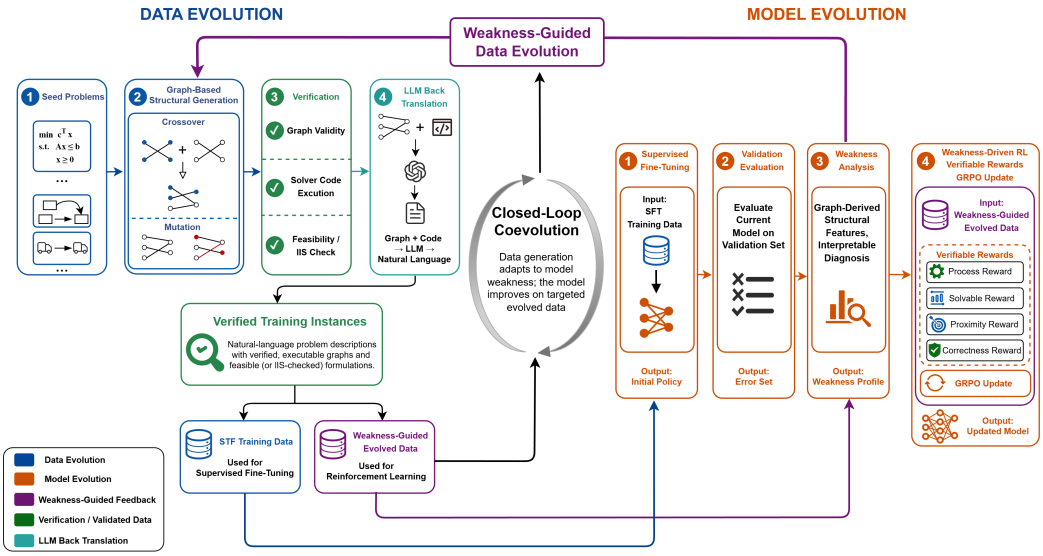

Figure 1: Weakness-Guided Data–Model Coevolution Framework.

We denote the algebraic formulation by $\mathcal{P} = (\min / \max, \mathbf{c}, \mathbf{A}, \mathbf{b}, \bowtie, \mathbf{l}, \mathbf{u}, \mathcal{I}, \mathcal{R})$, and its solver-executable program (e.g., Gurobi code) by $\mathcal{V}(\mathcal{P})$.

To support both structured generation and failure analysis, we represent each MILP as an attributed bipartite graph $\mathcal{G} = (V, E, \mathcal{A})$, following the variable–constraint graph representation widely used in learning-based MILP research [Gasse et al., 2019, Fan et al., 2023]. Specifically: (1) Nodes: $V = V_{\text{con}} \cup V_{\text{var}}$, where $V_{\text{con}} = \{s_1, \ldots, s_m, s_{\text{obj}}\}$ contains one node per constraint together with a dedicated objective node, and $V_{\text{var}} = \{x_1, \ldots, x_n\}$ contains one node per decision variable. (2) Edges: $E \subseteq V_{\text{con}} \times V_{\text{var}}$; an edge (s_i, x_j) is present when variable x_j appears in constraint s_i or in the objective with a nonzero coefficient, and the edge weight stores that coefficient. (3) Attributes: A constraint node stores its type, right-hand side, and relational sense; a variable node stores its lower bound, upper bound, and variable type. This graph serves as an internal representation for data generation and weakness diagnosis. It is *manipulable* (crossover and mutation operate directly on variables, constraints, coefficients, and substructures) and *analyzable* (graph statistics summarize structural regimes where the model succeeds or fails). A valid graph has exactly one objective node, complete variable metadata, and no isolated components; from any valid $\mathcal{G}$ we deterministically recover $\mathcal{P}$ and $\mathcal{V}(\mathcal{P})$.

The learning problem is therefore to estimate a mapping $f_\theta : \mathcal{Q} \to \mathcal{V}$ from natural-language descriptions to solver-executable programs. EvoOptiGraph improves this mapping indirectly by controlling the structure of the training data and adapting that structure over time.

3.2 Controllable Data Generation via Graph Evolution

We generate training instances by evolving graphs rather than only perturbing textual descriptions or numeric parameters. The initial population is built from 53 expert-designed seed generators from OptMATH [Lu et al., 2025], each corresponding to a classical problem family such as transportation, knapsack, or facility location. Sampling the parameters of these generators yields an initial dataset $\mathcal{D}_0 = \{(q_i, \mathcal{G}_i, c_i)\}_{i=1}^{N_0}$, where q_i is a natural-language description, $\mathcal{G}_i$ is the graph representation of the optimization instance, and $c_i = \mathcal{V}(\mathcal{P}_i)$ is the executable solver program compiled from the graph.

Genetic operations with structural constraints. We apply genetic operators directly to $\mathcal{G}$ subject to validity constraints. **Crossover** exchanges compatible subgraphs between two parent instances. In the default setting, crossover is performed within the same problem class to preserve semantic coherence; a smaller number of cross-class operations is allowed to increase structural diversity. The objective node is handled separately so that each offspring still contains exactly one objective. **Mutation** acts at three levels. *Parameter mutation* perturbs numerical quantities such as coefficients, bounds, and right-hand sides. *Structural mutation* edits local graph topology by inserting, deleting, or modifying variables or constraints while preserving linearity and metadata consistency. *Scale*

mutation changes problem size, e.g., expanding supply/demand nodes in a transportation instance. Every offspring is filtered by deterministic validity checks before entering the candidate pool.

Figure 2 shows an example: crossing a production scheduling problem (A) with a knapsack (B) yields hybrid (C); mutating (C)'s knapsack capacity (100→90) gives (C'). See appendix for full details.

Quality control and fitness function. Each candidate instance is verified before entering the training pool. The first stage checks graph structural validity. The second stage compiles the graph into Gurobi code and executes it. For feasible instances, we record solver statistics such as solve time and branch-and-bound nodes as difficulty metadata. For infeasible instances, we compute an Irreducible Infeasible Subset (IIS) [Gurobi Optimization, LLC, 2024, Chinneck, 2008]; instances with non-empty IIS are retained because they still define meaningful optimization structures, while candidates that fail to compile or execute are discarded. Approximately 3% infeasible examples are retained only in SFT and are excluded from weakness analysis and RL, so RL examples retain defined ground-truth objectives. Structural validity and solver feasibility rates are reported in Appendix C.4. Verification metadata is cached to avoid redundant solver calls. The fitness function balances three normalized objectives (min-max scaling to $[0, 1]$):

$$f(\mathcal{G}) = \beta \tilde{f}_{\text{diff}} + \gamma \tilde{f}_{\text{div}} + \delta \tilde{f}_{\text{weak}}.$$

Difficulty $\tilde{f}_{\text{diff}}$ is derived from solver statistics and favors nontrivial instances. Diversity $\tilde{f}_{\text{div}}$ is the average graph edit distance (GED) [Gao et al., 2010, Gasse et al., 2019, Fan et al., 2023] to the current population, discouraging collapse to a few templates. Weakness alignment $\tilde{f}_{\text{weak}}$ is the cosine similarity between the candidate's structural feature vector $\phi(\mathcal{G})$ and the current model weakness vector $\mathbf{w}$ (Section 3.3). This term is inactive initially ($\delta = 0$) and becomes important once the model has enough validation evidence to estimate its failure modes.

Back translation to natural language. For each verified graph, we prompt an LLM agent with its serialization and provenance to generate a natural-language description. A three-stage verification ensures quality: (1) rule-based checks (numerical consistency, coverage of variables and constraints, fidelity to objective and operator types); (2) the agent checks logical compatibility, regenerating up to a fixed budget if needed; (3) periodic value-matching audits on sampled instances (reconstruct solver code via a separate forward-modeling system and compare outcomes). Filter rates and the residual mismatch after filtering are reported in Appendix C.2.

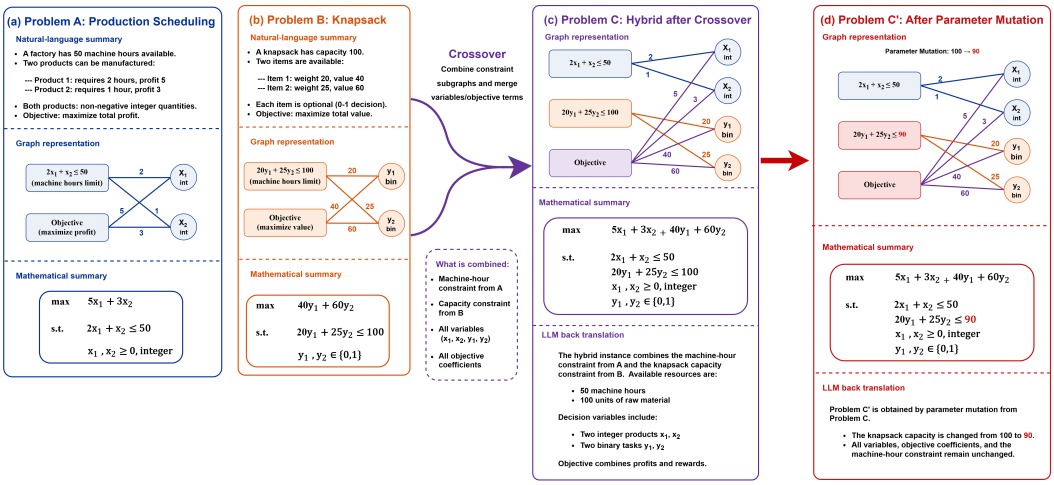

Figure 2: Crossover and mutation example. (a) Production scheduling; (b) knapsack; (c) hybrid after crossover (C); (d) mutated from C (C', capacity 100→90). Red highlight indicates mutation.

3.3 Two-Stage Model Training and Coevolution Loop

The training loop alternates between model fitting and data evolution: the current model is evaluated on a held-out validation set, and its failure patterns are summarized as a weakness vector $\mathbf{w}$, which biases the generator toward informative instances for RL. Algorithm 1 outlines the procedure.

Algorithm 1 EvoOptiGraph Training Loop

Input: Seed generators $\mathcal{S}$, base LLM with parameters θ_0, validation set $\mathcal{V}_{\text{val}} = \{(\mathcal{G}_i, y_i)\}_{i=1}^{N_{\text{val}}}$, number of iterations T
Output: Optimized model parameters θ_T
$\mathcal{D}_0 \leftarrow \text{EvoGenerate}(\mathcal{S})$ ▷ initial population
$\theta_{\text{warm}} \leftarrow \arg\min_\theta \mathcal{L}_{\text{SFT}}(\theta; \mathcal{D}_0)$ ▷ warm-up SFT
$\theta_0 \leftarrow \theta_{\text{warm}}$
for $t = 1$ to T **do**
  $\mathcal{E}_t \leftarrow \{(\mathcal{G}, y) \in \mathcal{V}_{\text{val}} \mid M_{\theta_{t-1}}(\mathcal{G}) \neq y\}$ ▷ error set
  $\Phi_t \leftarrow \{\phi(\mathcal{G}) \mid \mathcal{G} \in \mathcal{V}_{\text{val}}\}$ ▷ feature vectors
  $\mathbf{w}_t \leftarrow \text{SHAP}\big(\text{XGBoost}(\{(\phi(\mathcal{G}), \mathbf{1}_{(\mathcal{G},y)\in\mathcal{E}_t})\})\big)$
  Evolve new data using fitness: $f(\mathcal{G}) = \beta\tilde{f}_{\text{diff}} + \gamma\tilde{f}_{\text{div}} + \delta\tilde{f}_{\text{weak}}$
  $\mathcal{D}_{\text{RL}} \leftarrow \text{EvoGenerate}(\mathcal{S}, f(\mathcal{G}), \mathbf{w}_t)$ ▷ weakness-targeted data
  Sample candidate responses $\{y_i\}_{i=1}^{G} \sim \pi_{\theta_{t-1}}(\cdot \mid q)$ for each $(q, f_{\text{opt}}) \in \mathcal{D}_{\text{RL}}$
  Extract code blocks $\{c_i\}_{i=1}^{G}$ from $\{y_i\}_{i=1}^{G}$ and execute them in parallel
  Compute staged rewards $\{R_i\}_{i=1}^{G}$ using $R_{\text{process}} + R_{\text{solvable}} + R_{\text{proximity}} + R_{\text{correctness}}$
  $\theta_t \leftarrow \arg\max_\theta J_{\text{GRPO}}(\theta; \theta_{t-1}, \mathcal{D}_{\text{RL}}, \{R_i\})$ ▷ RL update
end for
return θ_T

Warm-up supervised fine-tuning. We train a base LLM on the initial dataset $\mathcal{D}_0$ by standard next-token prediction: $\mathcal{L}_{\text{SFT}} = -\sum_t \log P_\theta(c_t \mid c_{<t}, q)$, where $\mathcal{D}_0$ is generated from seed problems via graph evolution, q is the problem description and c_t is the t-th token of the solver code, yielding an initial policy M_{warm} competent for error analysis.

Structured weakness analysis. In each coevolution round, we evaluate the current model on a held-out validation set whose instances are paired with graphs and ground-truth outcomes. For every graph $\mathcal{G}$, we compute a structural feature vector $\phi(\mathcal{G})$ that includes size statistics (numbers of variables and constraints), type information (e.g., proportions of binary and integer variables), sparsity and density measures, graph-topological features (such as average degree and spectral quantities), and coefficient statistics (see Appendix for a detailed list). These features provide a low-dimensional summary of the optimization. We train an XGBoost classifier [Chen and Guestrin, 2016] to predict whether the model solves an instance correctly from $\phi(\mathcal{G})$. To extract interpretable feature contributions, we compute SHAP values [Lundberg and Lee, 2017] on the error cases. The weakness weight for feature k is $w_k = \frac{1}{|\mathcal{E}|}\sum_{i\in\mathcal{E}} \text{SHAP}_{i,k}$, where $\mathcal{E}$ is the set of validation instances answered incorrectly by the current model. The resulting vector $\mathbf{w} = [w_1, \ldots, w_d]$ summarizes structural properties most associated with failure and serves as a practical prioritization signal, not a causal statement.

Weakness-driven reinforcement learning. Using $\mathbf{w}$, we rerun the evolutionary generator and activate the weakness-alignment term $\tilde{f}_{\text{weak}} = \text{cosine_similarity}(\phi(\mathcal{G}), \mathbf{w})$, which biases evolution toward instances whose structure resembles the model's current failure regime. We then generate a new dataset $\mathcal{D}_{\text{RL}}$ by evolving instances with this weakness-aligned fitness for RL. Appendix E.2 details how $\mathbf{w}_t$ changes which candidates survive tournament selection.

We formulate the update as reinforcement learning with verifiable rewards (RLVR) [Lambert et al., 2024, Ye et al., 2025, Jiang et al., 2026]. Each training example comprises a description q and its ground-truth optimal objective value f_{opt}; the corresponding graph and solver code are used offline. During RL, the model generates a response, we extract and execute its code with Gurobi. The reward is a multi-stage sum:

$$R(c) = R_{\text{process}}(c) + R_{\text{solvable}}(c) + R_{\text{proximity}}(c) + R_{\text{correctness}}(c),$$

where $R_{\text{process}}(c) = 0.2$ if code executes without Python error, else 0. Let $\text{relerr} := \frac{|f_{\text{model}} - f_{\text{opt}}|}{|f_{\text{opt}}| + \delta}$ denote the relative error between the solver's returned objective f_{model} and the ground-truth objective value f_{opt}, with $\delta = 10^{-8}$. If the solver returns a feasible solution, we set $R_{\text{solvable}} = 0.1$ and $R_{\text{proximity}} = 0.2 \cdot \max(0, 1 - \text{relerr}/0.1)$; otherwise both are 0. Thus $R_{\text{proximity}}$ decays linearly from 0.2 at $\text{relerr} = 0$ to 0 at $\text{relerr} = 0.1$ (10% relative error). Intuitively, a solution whose relative error

Table 1: The overall performance of `EvoOptiGraph` and baselines with Pass@1 accuracy (%) on six OR benchmarks. Instance counts are NL4OPT 214, MAMO-EasyLP 545, MAMO-ComplexLP 111, NLP4LP 178, ComplexOR 18, and IndustryOR 42. Scores marked with * are reference-only results on uncleaned splits (Appendix A.1) and are excluded from the ranking; missing entries are denoted with (-). Among cleaned-split methods, the best result in each column is **bold** and the second-best is underlined; tied values share the same mark.

Model	Params	NL4OPT	MAMO		NLP4LP	CompOR	IndOR	Avg.
			EasyLP	ComplexLP				
Zero-shot LLMs								
`GPT-4o`	Closed	73.4	95.2	46.0	87.6	38.9	66.7	68.0
`DeepSeek-V3`	671B	79.9	96.7	55.0	94.9	44.4	**69.1**	73.3
`Qwen3-32B`	32B	76.1	96.9	32.4	90.5	38.9	57.1	65.3
`Qwen2.5-72B-Inst`	72B	79.9	95.8	33.3	89.9	38.9	61.9	66.6
Agentic Methods								
`OptiMUS-v0.3`	Closed	79.8	92.4	52.1	89.8	52.2	54.3	70.1
`CoT`	Closed	62.2	49.5	42.3	74.7	39.2	40.5	51.4
`CoE`	Closed	66.7	94.4	50.6	87.4	**57.1**	31.2	64.6
Fine-tuned LLMs								
`ORLM`	8B	73.8	90.4	59.5	76.4	50.0	42.9	65.5
`LLMOPT` (origin)	14B	80.3*	89.5*	44.1*	73.4*	35.3*	29.0*	58.6*
`OptMATH` (origin)	32B	95.9*	89.9*	54.1*	-	-	-	-
`EvoOptiGraph`	8B	**96.7**	**98.0**	**69.4**	**97.2**	50.0	57.1	**78.1**

Abbreviations: CompOR: ComplexOR, IndOR: IndustryOR, Avg: Macro-Average, Inst: Instruct.

is within 10% of the ground truth is more likely to capture the main structure (e.g., minor coefficient errors), while a large deviation tends to signal a fundamental mistake. The correctness bonus is $R_{\text{correctness}} = 0.5$ when solver status is *optimal* and $\text{relerr} \le \tau_{\text{rel}}$ ($\tau_{\text{rel}} = 10^{-4}$), else 0. Total reward in $[0, 1]$. This layered design encourages runnable code, any feasible solution, numerical closeness (within 10%), and exact matches.

To optimize the policy π_θ, we adopt group-relative policy optimization (GRPO) [Shao et al., 2024, Guo et al., 2025]. For each prompt q, we sample N candidate responses $\{y_i\}_{i=1}^N$ from the current policy $\pi_{\theta_{\text{old}}}$, evaluate their rewards $\{R_i\}_{i=1}^N$, and normalize them within the group to obtain advantages $\hat{A}_i = (R_i - \mu_R)/\sigma_R$. We then maximize

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{q\sim\mathcal{D}_{\text{RL}},\,\{y_i\}\sim\pi_{\theta_{\text{old}}}}\left[\frac{1}{N}\sum_{i=1}^{N}\left(\min\left(r_i(\theta)\hat{A}_i,\ \text{clip}(r_i(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_i\right) - \widetilde{\beta}\cdot\mathbb{D}_{\text{KL}}(\pi_\theta\|\pi_{\text{ref}})\right)\right],$$

where $r_i(\theta) = \frac{\pi_\theta(y_i|q)}{\pi_{\theta_{\text{old}}}(y_i|q)}$ is the importance ratio, ϵ is the clipping hyperparameter, $\widetilde{\beta}$ controls the KL penalty strength, and π_{ref} is the SFT reference policy. The KL term regularizes the RL update so that the policy does not drift too far from the supervised initialization while still exploiting solver-based rewards.

Iterative coevolution loop. At each iteration t, we first diagnose the current model M_t on the validation set to obtain the weakness vector $\mathbf{w}_t$. We then evolve new instances using weakness-aligned fitness to generate $\mathcal{D}_{\text{RL}}$, and update M_t to M_{t+1} via GRPO. This closed loop makes the data distribution co-evolve with the model, generating new data in response to observed errors rather than statically. Figure 1 shows our overall framework diagram.

4 Experiments

4.1 Experiment Setup

Data Generation. We generate training data via genetic graph evolution. Starting from 53 expert-designed seed problem families Lu et al. [2025], the generator applies mutation and crossover operators to produce new optimization instances. Beyond the original 53 families, we additionally construct 30 cross-family hybrid categories through inter-class crossover. These hybrid categories

make up about 20% of the data generated in each round. In terms of scale, we use 5,000 generated instances for the initial SFT stage. After warm-up, each RL iteration uses a newly generated pool of 1,000 instances. We further generate 2,000 additional instances per iteration for structured weakness analysis, which are used to train the XGBoost classifier.

Benchmarks. Our evaluation is conducted on six public benchmarks commonly used in the OR community: NL4Opt Ramamonjison et al. [2023], MAMO, which is divided into EasyLP and ComplexLP Huang et al. [2024], NLP4LP AhmadiTeshnizi et al. [2024], ComplexOR Xiao et al. [2023], IndustryOR Huang et al. [2025]. See Appendix for more details about these benchmarks.

Baseline. To ensure a thorough comparison, we benchmark against a comprehensive set of baselines spanning three categories: zero-shot generalist LLMs, specialized fine-tuned LLMs, and agentic methods. The zero-shot generalist LLMs include GPT-4o OpenAI [2026], DeepSeek-V3 Liu et al. [2024a], Qwen3-32B Yang et al. [2025], and Qwen2.5-72B-Instruct Qwen et al. [2025]. The specialized fine-tuned LLMs include ORLM (8B) Huang et al. [2025], LLMOPT (14B) Jiang et al. [2024], and OptMATH (32B) Lu et al. [2025]. The agentic methods include multi-step reasoning frameworks that orchestrate LLMs to solve problems, such as OptiMUS-v0.3 AhmadiTeshnizi et al. [2024], Chain-of-Thought (CoT) Wei et al. [2022] and Chain-of-Experts (CoE) Xiao et al. [2023]. For OptiMUS-v0.3, we use the results reported by Zhang et al. [2026], while for CoE, CoT, and ORLM, we use the results reported by Xiao et al. [2025], as their evaluations are conducted on the same dataset as ours, namely the cleaned version released by Xiao et al. [2025]. For OptMATH and LLMOPT, marked with * in Table 1, we report reference results on the original uncleaned splits. Exact sources and the single-shot protocol used for LLMOPT are given in Appendix A.1.

Implementation Details. In the back-translation stage of data generation, we use DeepSeek-V3 Liu et al. [2024a] to generate natural-language descriptions from verified graph instances. We adopt the prompting framework of OptMATH Lu et al. [2025], which consists of three stages—generate, check, and regenerate—with full prompt details provided in the appendix. We develop EvoOptiGraph by fine-tuning the publicly available Qwen3-8B Yang et al. [2025] model. Detailed configurations and procedures of the fine-tuning process are also deferred to the appendix. As the primary evaluation metric, we adopt pass@1 accuracy, which measures whether the optimal objective value obtained from the generated optimization code matches the reference value provided by the benchmark.

4.2 Main Results

Table 1 presents the overall Pass@1 results on six OR benchmarks. Among methods evaluated on the cleaned splits, EvoOptiGraph achieves the best macro-average Pass@1, 78.1%. Mean and standard deviation over 2 training seeds and 3 evaluation runs are reported in Appendix A.3. It consistently outperforms zero-shot LLMs, agentic methods, and prior fine-tuned models, showing the effectiveness of our two-stage model training and coevolution loop. Across individual benchmarks, EvoOptiGraph achieves the strongest or tied-best results on nearly all tasks, demonstrating robust generalization across diverse optimization problem types. Notably, this performance is achieved with an 8B backbone, yet it still surpasses substantially larger models. This suggests that, for optimization-oriented code generation, targeted data evolution and structured weakness adaptation can be more effective than simply increasing model scale.

4.3 Data diversity comparison

We evaluate data diversity using the Structural Similarity metric introduced in GenBench-MILP Luo et al. [2026]. Specifically, we compare the similarity between the generated instances and the 53 original problem types for two data generation methods: OptMATH Lu et al. [2025] and our method. Under this metric, lower similarity indicates higher diversity. For each method, we first generate 5,000 instances in total. Then, to ensure a statistically meaningful comparison, we randomly sample 212 instances at a time, compute their structural similarity to the 53 original problem types, and repeat this process 50 times. We report the average similarity over these 50 runs. The results show that the instances generated by our method achieve a structural similarity of 66.93%, whereas those generated by OptMATH achieve 93.04%. This substantial gap indicates that our method produces significantly more diverse data. A possible reason is that OptMATH mainly perturbs the explicit parameters of the original problems, while our method modifies the underlying problem structure itself, leading to richer structural variations.

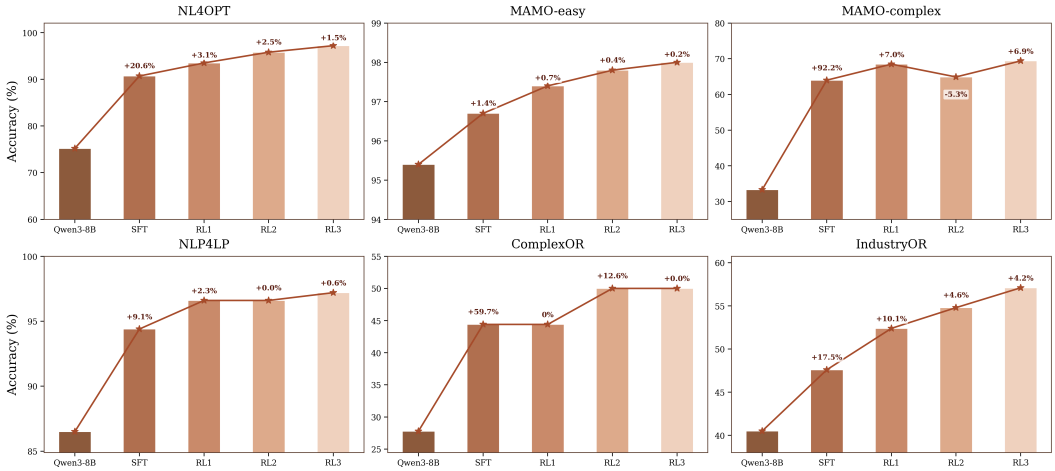

Figure 3: Accuracy trajectory across the coevolution loop. The relative improvement of current iteration over the previous one is demonstrated on the corresponding bar.

Table 2: Ablation study with Pass@1 accuracy (%) on six OR benchmarks.

Type	NL4OPT	MAMO		NLP4LP	CompOR	IndOR	Avg.
		EasyLP	ComplexLP				
`All-SFT`	85.5	96.0	55.9	91.6	38.9	42.9	68.5 (↓ **9.6**)
`All-RL`	72.0	96.3	33.3	87.6	38.9	52.4	63.4 (↓ **14.7**)

4.4 Analysis of Coevolution Loop

We analyze the coevolution loop by tracking model performance from the base model to the warm-up SFT stage and three subsequent reinforcement-learning rounds. Figure 3 reveals three observations. First, the warm-up SFT stage provides the largest foundational improvement across all benchmarks, especially on structurally challenging datasets such as MAMO-complex and ComplexOR, indicating that supervised training on verified optimization data is critical for establishing basic optimization modeling and code-generation competence. Second, the subsequent coevolution rounds consistently build on top of the SFT model on most benchmarks, yielding steady additional gains. This trend suggests that the weakness-guided generate-and-train loop continues to uncover informative failure modes even after supervised learning has saturated, rather than merely repeating the same data distribution. We also observe a non-monotonic trajectory on MAMO-complex, where performance drops at RL2 before recovering at RL3, which suggests that adaptive data evolution may temporarily overspecialize to certain failure regimes but remains self-correcting over subsequent rounds.

4.5 Ablation Study

To investigate the effectiveness of the two-stage training scheme and the coevolution loop, we conduct an ablation study with two variants: SFT-only and RL-only. Compared with SFT-only and RL-only, EvoOptiGraph improves the average accuracy by 9.6% and 14.7%, respectively. Detailed results are presented in the last row of Table 1 and Table 2. We find that SFT-only brings clear gains in the first round, but its improvement soon saturates and can even deteriorate in later rounds. One possible explanation is that repeated supervised updates on synthetic data may cause the model to overfit to recurring data patterns, while failing to further improve outcome-level optimization correctness. In contrast, RL-only shows only limited improvement, likely because without an SFT warm-up, the model does not have a sufficiently strong initialization, making reward-driven learning unstable and inefficient. For the accuracy trajectory, please refer to Figures 4 and 5 in the appendix.

To isolate the contribution of weakness-driven targeting, we compare three instance-selection strategies under the same SFT warm-up, GRPO recipe, and per-round data budget (Table 3). Full fitness uses weakness alignment, with δ following the schedule in Appendix C.3. The $\delta = 0$ variant keeps only the difficulty and diversity terms. Uniform sampling draws evolved instances with equal probability. Enabling weakness alignment improves the macro-average Pass@1 by 10.8% over $\delta = 0$ (78.1% vs. 67.3%). Graph-based evolution with difficulty and diversity filtering still outperforms

Table 3: Ablation of weakness alignment with Pass@1 accuracy (%) on six OR benchmarks.

Setting	NL4OPT	MAMO		NLP4LP	CompOR	IndOR	Avg.
		EasyLP	ComplexLP				
(a) Full fitness	96.7	98.0	69.4	97.2	50.0	57.1	78.1
(b) $\delta = 0$	87.4	95.4	45.0	91.6	38.9	45.2	67.3
(c) Uniform	83.2	94.7	41.4	89.9	38.9	40.4	64.8

uniform sampling by 2.5% (67.3% vs. 64.8%). Structural evolution is therefore beneficial on its own, and aligning new instances with the model's failure distribution supplies a further, larger gain.

5 Conclusion and Future Work

We presented EvoOptiGraph, a weakness-driven coevolution framework of data and models for natural-language-to-optimization modeling. By representing MILPs as attributed bipartite graphs, it unifies structural data generation, failure diagnosis, and model adaptation in a closed loop, combining supervised fine-tuning with reinforcement learning under verifiable solver-based rewards. Experiments on multiple public benchmarks show consistent gains over baselines; our 8B model achieves the best macro-average Pass@1, demonstrating that targeted structural data evolution can be more effective than scaling model size alone. Stage-wise and ablation results confirm that weakness-driven coevolution yields additional gains beyond SFT or RL alone.

Several limitations remain: the current framework is limited to MILPs and the coverage of available seed generators, verification becomes expensive for larger instances, and the weakness vector is a practical prioritization signal rather than a causal account. Future work includes extending EvoOptiGraph to broader optimization classes, improving efficiency, and studying transferability of weakness signals across problem families. We hope this work offers a useful step toward adaptive training pipelines for formal reasoning and optimization with large language models.

References

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. Optimus: Scalable optimization modeling with (mi) lp solvers and large language models. *arXiv preprint arXiv:2402.10172*, 2024.

Hossein Alipour, Mario Andrés Muñoz, and Kate Smith-Miles. Enhanced instance space analysis for the maximum flow problem. *European Journal of Operational Research*, 304(2):411–428, 2023.

Andreas Antoniou and Wu-Sheng Lu. *Practical optimization: algorithms and engineering applications*. Springer, 2007.

Nicolás Astorga, Tennison Liu, Yuanzhang Xiao, and Mihaela Van Der Schaar. Autoformulation of mathematical optimization models using llms. *arXiv preprint arXiv:2411.01679*, 2024.

David B Blumenthal, Nicolas Boria, Johann Gamper, Sébastien Bougleux, and Luc Brun. Comparing heuristics for graph edit distance computation: Db blumenthal et al. *The VLDB journal*, 29(1): 419–458, 2020.

Simon Bowly. *Stress testing mixed integer programming solvers through new test instance generation methods.* PhD thesis, University of Melbourne, Parkville, Victoria, Australia, 2019.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. Solver-informed rl: Grounding large language models for authentic optimization modeling. *arXiv preprint arXiv:2505.11792*, 2025.

John W Chinneck. *Feasibility and infeasibility in optimization: algorithms and computational methods*. Springer, 2008.

Ning Dai, Zheng Wu, Renjie Zheng, Ziyun Wei, Wenlei Shi, Xing Jin, Guanlin Liu, Chen Dun, Liang Huang, and Lin Yan. Process supervision-guided policy optimization for code generation. *arXiv preprint arXiv:2410.17621*, 2024.

Zhenan Fan, Xinglu Wang, Oleksandr Yakovenko, Abdullah Ali Sivas, Owen Ren, Yong Zhang, and Zirui Zhou. Smart initial basis selection for linear programs. In *International Conference on Machine Learning*, pages 9650–9664. PMLR, 2023.

Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010.

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.

Zijie Geng, Xijun Li, Jie Wang, Xiao Li, Yongdong Zhang, and Feng Wu. A deep instance generative framework for milp solvers under limited data availability. *Advances in Neural Information Processing Systems*, 36:26025–26047, 2023.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL `https://www.gurobi.com`.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. Orlm: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 73(6):2986–3009, 2025.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. Mamo: a mathematical modeling benchmark with solvers. *arXiv preprint arXiv:2405.13144*, 2024.

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. Llmopt: Learning to define and solve general optimization problems from scratch. *arXiv preprint arXiv:2410.13213*, 2024.

Yuxin Jiang, Yufei Wang, Qiyuan Zhang, Xingshan Zeng, Liangyou Li, Jierun Chen, Chaofan Tao, Haoli Bai, and Lifeng Shang. From verifiable dot to reward chain: Harnessing verifiable reference-based rewards for reinforcement learning of open-ended generation. *arXiv preprint arXiv:2601.18533*, 2026.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Sirui Li, Janardhan Kulkarni, Ishai Menache, Cathy Wu, and Beibin Li. Towards foundation models for mixed integer linear programming. *arXiv preprint arXiv:2410.08288*, 2024.

Zhong Li, Hongliang Lu, Tao Wei, Wenyu Liu, Yuxuan Chen, Yuan Lan, Fan Zhang, and Zaiwen Wen. Constructing industrial-scale optimization modeling benchmark. *arXiv preprint arXiv:2602.10450*, 2026.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024a.

Chang Liu, Kate Smith-Miles, Tony Wauters, and Alysson M Costa. Instance space analysis for 2d bin packing mathematical models. *European Journal of Operational Research*, 315(2):484–498, 2024b.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *arXiv preprint arXiv:2401.02051*, 2024c.

Haoyang Liu, Jie Wang, Wanbo Zhang, Zijie Geng, Yufei Kuang, Xijun Li, Bin Li, Yongdong Zhang, and Feng Wu. Milp-studio: Milp instance generation via block structure decomposition. *Advances in Neural Information Processing Systems*, 37:54357–54395, 2024d.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. Optmath: A scalable bidirectional data synthesis framework for optimization modeling. *arXiv preprint arXiv:2502.11102*, 2025.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

Yidong Luo, Chenguang Wang, Dong Li, and Tianshu Yu. Are your generated instances truly useful? genbench-milp: A benchmark suite for milp instance generation, 2026. URL `https://arxiv.org/abs/2505.24779`.

OpenAI. Chatgpt (GPT-4o version) [large language model], 2026. URL `https://chat.openai.com`. Accessed: 2026-04-29.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL `https://arxiv.org/abs/2412.15115`.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, et al. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 Competition Track*, pages 189–203. PMLR, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Ajay Singh. An overview of the optimization modelling applications. *Journal of Hydrology*, 466: 167–182, 2012.

Kate Smith-Miles and Simon Bowly. Generating new test instances by evolving in instance space. *Computers & Operations Research*, 63:102–113, 2015.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformers Reinforcement Learning, 2020. URL `https://github.com/huggingface/trl`.

Teng Wang, Wing-Yin Yu, Zhenqi He, Zehua Liu, Hailei Gong, Han Wu, Xiongwei Han, Wei Shi, Ruifeng She, Fangzhou Zhu, and Tao Zhong. Bpp-search: Enhancing tree of thought reasoning for mathematical modeling problem solving, 2025. URL `https://arxiv.org/abs/2411.17404`.

Yilin Wang, Heng Zhou, Dongxing Mao, Linjie Li, Jingru Tan, Haochen Han, Zhengyuan Yang, Alex Jinpeng Wang, and Min Li. Or-prm: A process reward model for algorithmic problem in operations research. In *The Fourteenth International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, et al. Chain-of-experts: When llms meet complex operations research problems. In *The Twelfth International Conference on Learning Representations*, 2023.

Ziyang Xiao, Jingrong Xie, Lilin Xu, Shisi Guan, Jingyan Zhu, Xiongwei Han, Xiaojin Fu, WingYin Yu, Han Wu, Wei Shi, et al. A survey of optimization modeling meets llms: Progress and future directions. *arXiv preprint arXiv:2508.10047*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Angen Ye, Zeyu Zhang, Boyuan Wang, Xiaofeng Wang, Dapeng Zhang, and Zheng Zhu. Vla-r1: Enhancing reasoning in vision-language-action models. *arXiv preprint arXiv:2510.01623*, 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.

Yansen Zhang, Qingcan Kang, Yujie Chen, Yufei Wang, Xiongwei Han, Tao Zhong, Mingxuan Yuan, and Chen Ma. Sac-opt: Semantic anchors for iterative correction in optimization modeling, 2026. URL `https://arxiv.org/abs/2510.05115`.

Congmin Zheng, Jiachen Zhu, Jianghao Lin, Xinyi Dai, Yong Yu, Weinan Zhang, and Mengyue Yang. Cold: Counterfactually-guided length debiasing for process reward models. *arXiv preprint arXiv:2507.15698*, 2025a.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 3: system demonstrations)*, pages 400–410, 2024.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in llm-based automatic heuristic design. *arXiv preprint arXiv:2501.08603*, 2025b.

Jiachen Zhu, Congmin Zheng, Jianghao Lin, Kounianhua Du, Ying Wen, Yong Yu, Jun Wang, and Weinan Zhang. Retrieval-augmented process reward model for generalizable mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8453–8468, 2025.

A Benchmarks

A large number of datasets and dataset-construction methods have been introduced to improve the capability of LLMs in solving OR problems, and these datasets have become increasingly complex and realistic AhmadiTeshnizi et al. [2024]. Xiao et al. [2025] provide a recent and comprehensive survey of benchmark datasets for natural-language-described optimization problems. Their study highlights several recurring issues in existing benchmarks, such as logical inconsistencies, ambiguous problem statements, and incorrect ground-truth answers, and further rectifies part of these errors.

NL4OPT The NL4Opt dataset, introduced by Ramamonjison et al. [2023] for the NeurIPS 2022 NL4Opt competition, is a benchmark containing approximately 1,100 annotated linear programming problems. It is designed to study the mapping from natural language descriptions to formal optimization models. The original release comprises 713 training instances, 99 validation instances, and 289 test instances. Due to the noise and errors in the original dataset, we follow Xiao et al. [2025] and use their cleaned version instead. Specifically, our experiments are conducted on 214 cleaned instances.

MAMO The MAMO dataset was introduced by Huang et al. [2024] as a benchmark for evaluating large language models on mathematical modeling tasks. It focuses on LP and MILP problems, with an emphasis on the formulation process rather than the solving step itself. The original dataset is divided into two subsets: Easy LP with 652 instances and Complex LP with 211 instances. In our work, rather than using the original release directly, we adopt the cleaned version provided by Xiao et al. After data cleaning, we use 545 easy instances and 111 complex instances in our experiments.

NLP4LP The NLP4LP benchmark was introduced by AhmadiTeshnizi et al. [2024] for evaluating the ability of large language models to translate natural language operations research problems into solver-ready code and mathematical formulations. It consists of 269 human-authored LP and MILP problems covering classical operations research domains such as scheduling, knapsack allocation, and production planning. In our work, rather than using the original release directly, we adopt the cleaned version provided by Xiao et al. After data cleaning, we use 178 instances in our experiments.

COMPLEXOR The ComplexOR dataset was introduced by Xiao et al. [2023] as a benchmark for evaluating large language models on challenging operations research modeling tasks. It consists of complex optimization problems drawn from advanced OR case studies and difficult problem classes, such as multi-step lot-sizing, intricate scheduling, and supply chain design. These instances are primarily large-scale or conceptually complex mixed-integer linear programming problems that require multi-step reasoning. In our work, we use 18 instances from this dataset in our experiments.

INDUSTRYOR The IndustryOR dataset was introduced by Huang et al. [2025] as a benchmark for evaluating large language models on real-world operations research problems. It consists of 100 optimization instances drawn from practical domains such as manufacturing, logistics, finance, and energy. The dataset covers multiple problem categories, including linear, integer, mixed-integer, nonlinear, and other types. In our work, rather than using the original release directly, we adopt the cleaned version provided by Xiao et al. After data cleaning, we use 42 instances in our experiments.

A.1 Baseline Provenance

Table 1 separates results on the cleaned splits from reference numbers on the original releases. Zero-shot LLMs, OptiMUS-v0.3, CoT, CoE, and ORLM are compared on the cleaned splits of Xiao et al. [2025], the same splits used for EvoOptiGraph, under a single-shot Pass@1 protocol with Gurobi verification. CoT, CoE, and ORLM follow the numbers reported by Xiao et al. [2025]. OptiMUS-v0.3 follows Zhang et al. [2026]. OptMATH Lu et al. [2025] and LLMOPT Jiang et al. [2024], marked with * in the table, were evaluated on the original uncleaned splits and are reported only as reference; they are not directly comparable to the cleaned-split results. Because those papers do not report matched numbers on the cleaned splits, differences relative to EvoOptiGraph confound method quality with dataset cleaning and should not be read as a head-to-head comparison. The LLMOPT numbers are taken from the official repository (`https://github.com/antgroup/LLMOPT`) under the Single-Agent setting without self-correction, which matches our single-shot Pass@1 protocol.

Table 4: Pass@1 of `EvoOptiGraph` (%). Each entry is the mean ± standard deviation over 3 evaluation runs and 2 training seeds. Avg. is the macro-average of the six per-benchmark means.

Model	Params	NL4OPT	MAMO		NLP4LP	CompOR	IndOR	Avg.
			EasyLP	ComplexLP				
`EvoOptiGraph`	8B	96.9±0.4	97.8±0.3	67.7±2.0	96.6±0.7	50.0±0.0	55.2±2.3	77.4

A.2 Evaluation Protocol and Remaining Caveats

Following common practice in natural-language-to-optimization modeling, we score Pass@1 by objective-value equality: a generated program is counted as correct when Gurobi returns an optimal objective that matches the benchmark reference within the relative-error tolerance $\tau_{\text{rel}} = 10^{-4}$ used in training. This protocol is solver-verifiable and consistent with the RL correctness reward, but it does not directly inspect the algebraic formulation. Two programs with different constraint sets can still share the same optimal value, so a match does not certify structural identity with the intended model. Stronger structure-aware verification is left to future work.

ComplexOR and IndustryOR contain 18 and 42 instances, respectively. On these two smallest splits, a single additional correct or incorrect prediction changes Pass@1 by about 5.6 and 2.4 percentage points. Scores on these benchmarks therefore remain sensitive to a few instances and are most informative when read together with the larger splits and the macro-average.

A.3 Mean and Standard Deviation

Table 1 reports point estimates of Pass@1. For EvoOptiGraph, we additionally measure variability by training with 2 random seeds and evaluating each trained model on 3 independent runs. Table 4 reports the mean ± standard deviation. Each entry is the mean over 3 evaluation runs and 2 training seeds. Avg. is the macro-average of the six per-benchmark means. The point estimates in Table 1 fall within one standard deviation of these means on every benchmark. Standard deviations are largest on MAMO-ComplexLP and IndustryOR; ComplexOR is unchanged across the six runs. Baseline entries in Table 1 remain the published point estimates.

B Detailed Problem Descriptions for Illustrative Examples

This appendix provides complete descriptions of the instances used in the crossover and mutation illustration (Figure 2) as well as additional examples generated by different crossover operations.

Problem A: Production Scheduling *Natural language description.* A factory has 50 machine hours available. Two products can be manufactured:

- Product 1 requires 2 hours per unit and yields a profit of 5 per unit.
- Product 2 requires 1 hour per unit and yields a profit of 3 per unit.

Both products can be produced in any non-negative integer quantity. The objective is to maximize total profit.

Mathematical formulation.

$$\begin{aligned} \max \quad & 5x_1 + 3x_2 \\ \text{s.t.} \quad & 2x_1 + x_2 \leq 50, \\ & x_1, x_2 \geq 0, \text{ integer.} \end{aligned}$$

Problem B: Knapsack *Natural language description.* A knapsack has a capacity of 100. Two items are available:

- Item 1 weighs 20 and has value 40.
- Item 2 weighs 25 and has value 60.

Each item can be selected at most once (0-1 decision). The objective is to maximize total value.

Mathematical formulation.

$$\begin{aligned} \max \quad & 40y_1 + 60y_2 \\ \text{s.t.} \quad & 20y_1 + 25y_2 \leq 100, \\ & y_1, y_2 \in \{0, 1\}. \end{aligned}$$

Problem C: Hybrid after Crossover *Natural language description.* This instance is generated by combining the constraint subgraphs of Problem A and Problem B. It inherits both constraints: the machine-hour limit from A and the knapsack capacity from B. The variables and objective coefficients are the union of those from A and B. In concrete terms, the factory has 50 machine hours and 100 units of raw material. Two integer products are available:

- Product 1 requires 2 hours and yields a profit of 5.
- Product 2 requires 1 hour and yields a profit of 3.

Two binary tasks are available:

- Task 1 consumes 20 units of material and gives a reward of 40.
- Task 2 consumes 25 units of material and gives a reward of 60.

Total profit is the sum of product profits and task rewards. The goal is to maximize total profit.

Mathematical formulation.

$$\begin{aligned} \max \quad & 5x_1 + 3x_2 + 40y_1 + 60y_2 \\ \text{s.t.} \quad & 2x_1 + x_2 \leq 50, \\ & 20y_1 + 25y_2 \leq 100, \\ & x_1, x_2 \geq 0, \text{ integer}, \\ & y_1, y_2 \in \{0, 1\}. \end{aligned}$$

Problem C': Mutation of Problem C *Natural language description.* Problem C' is derived from Problem C by a parameter mutation: the raw material capacity is reduced from 100 to 90. All other aspects (variables, constraints, objective coefficients) remain unchanged.

Mathematical formulation.

$$\begin{aligned} \max \quad & 5x_1 + 3x_2 + 40y_1 + 60y_2 \\ \text{s.t.} \quad & 2x_1 + x_2 \leq 50, \\ & 20y_1 + 25y_2 \leq 90, \\ & x_1, x_2 \geq 0, \text{ integer}, \\ & y_1, y_2 \in \{0, 1\}. \end{aligned}$$

More diverse instances via different crossover operations. Beyond the basic crossover shown in Problem C (combining entire constraint subgraphs), our framework supports other crossover variants that produce even more diverse structures. We illustrate two such variants below: Problem D demonstrates a simple exchange of constraint subgraphs between parent classes, while Problem E shows a more sophisticated crossover that creates a novel mixed constraint.

Problem D: Another Crossover Example *Natural language description.* This instance is generated by a different crossover operation: we exchange the entire constraint subgraph of Problem A with that of Problem B. The resulting offspring retains the variables and objective coefficients from A but adopts the constraint from B. In practical terms, the factory now has 100 units of raw material. Two integer products are available:

- Product 1 consumes 20 units of raw material and yields a profit of 5 per unit.
- Product 2 consumes 25 units of raw material and yields a profit of 3 per unit.

Both products can be produced in any non-negative integer quantity. The objective is to maximize total profit.

Mathematical formulation.

$$\begin{aligned} \max \quad & 5x_1 + 3x_2 \\ \text{s.t.} \quad & 20x_1 + 25x_2 \leq 100, \\ & x_1, x_2 \geq 0, \text{ integer}. \end{aligned}$$

Discussion. This operation produces a structurally distinct instance: a production scheduling problem transformed into a knapsack with integer items. It demonstrates the flexibility of our crossover to exchange subgraphs between different problem classes.

Problem E: Complex Crossover with a Novel Mixed Constraint *Parent descriptions (same as A and B).* We use the same parents A and B as above.

Crossover operation. We perform a three-step crossover:

1. Inherit the machine-hour constraint from Parent A.
2. Inherit the knapsack capacity constraint from Parent B.
3. Create a new constraint by combining coefficients from both parents: take the coefficient of x_1 from Parent A (2) and the coefficient of y_1 from Parent B (20), and form the inequality $2x_1 + 20y_1 \leq 30$, where the right-hand side is randomly generated (30). This new constraint couples an integer variable and a binary variable.

The objective is the sum of both parents' objectives.

Resulting Problem E: Multi-Constraint Hybrid. The offspring contains three constraints and a mix of integer and binary variables.

Natural language description. A factory has three independent resource limits:

- **Machine hours:** 50 hours available. Two integer products can be produced: Product 1 requires 2 hours and yields a profit of 5 per unit; Product 2 requires 1 hour and yields a profit of 3 per unit.
- **Raw material:** 100 units available. Two binary tasks can be selected: Task 1 consumes 20 units of material and gives a reward of 40; Task 2 consumes 25 units of material and gives a reward of 60.
- **Catalyst constraint (mixed coupling):** Producing Product 1 consumes a rare catalyst (2 units per product); activating Task 1 (e.g., starting an auxiliary reactor) consumes 20 units of the same catalyst. The daily supply of catalyst is limited to 30 units. Hence, the production of Product 1 and the activation of Task 1 must satisfy $2x_1 + 20y_1 \leq 30$.

The goal is to maximize total profit, which is the sum of product profits and task rewards.

Mathematical formulation.

$$\begin{aligned} \max \quad & 5x_1 + 3x_2 + 40y_1 + 60y_2 \\ \text{s.t.} \quad & 2x_1 + x_2 \leq 50, \\ & 20y_1 + 25y_2 \leq 100, \\ & 2x_1 + 20y_1 \leq 30, \\ & x_1, x_2 \geq 0, \text{ integer}, \\ & y_1, y_2 \in \{0, 1\}. \end{aligned}$$

Discussion. This example demonstrates a highly flexible crossover that not only inherits complete constraints from parents but also creates entirely new constraints by mixing coefficients and variable types. The resulting instance has a richer structure (three constraints, mixed variable domains, a coupling between an integer and a binary variable) that would be impossible to obtain by simple parameter perturbation or by merely concatenating parent constraints. Such complex crossovers are crucial for generating structurally diverse training data, enabling the model to learn intricate trade-offs between different resource types and decision variables. Our framework supports this kind of compositional crossover by allowing arbitrary compatible subgraphs (including partial coefficient vectors) to be exchanged or combined.

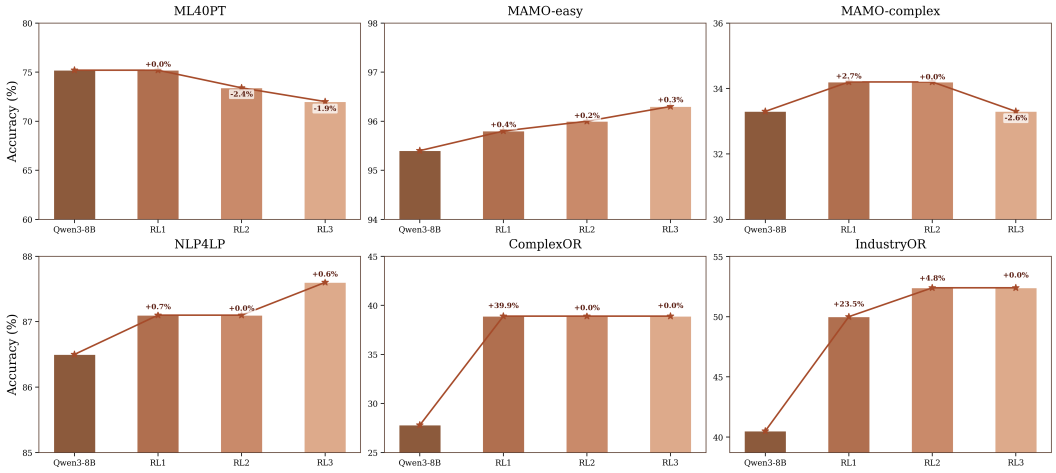

Figure 4: Accuracy trajectory of RL-only. The relative improvement of current iteration over the previous one is demonstrated on the corresponding bar.

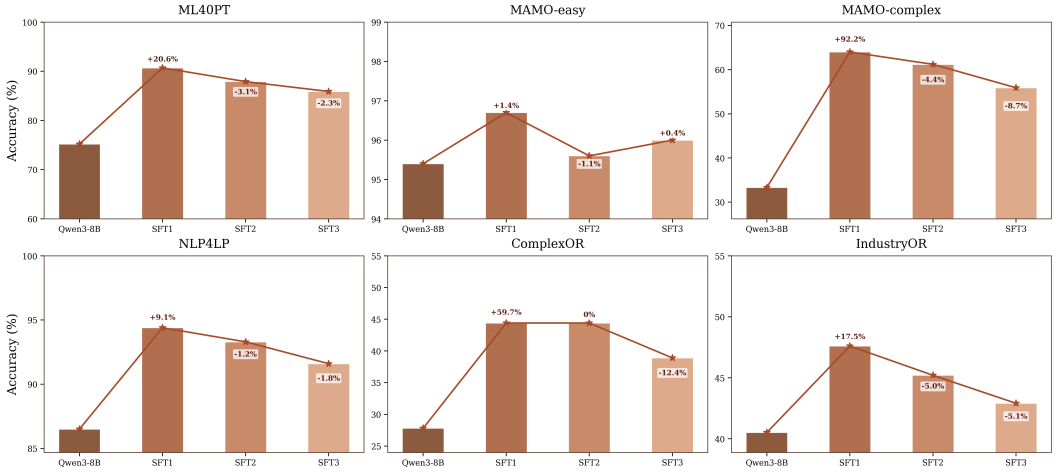

Figure 5: Accuracy trajectory of SFT-only. The relative improvement of current iteration over the previous one is demonstrated on the corresponding bar.

C Details of Implementation

C.1 SFT & RL

For supervised fine-tuning (SFT), we use the LLAMAFACTORY framework Zheng et al. [2024] with a learning rate of 1×10^{-4} for 3–5 epochs. We employ LoRA with rank 32, alpha 64, and dropout 0.05. The warm-up ratio is set to 0.1, and the optimizer is `adamw_8bit`. Although some hyperparameters vary slightly across experiments, the overall setup remains similar. For reinforcement learning (RL), we use the GRPOTRAINER implementation from the TRL library von Werra et al. [2020]. Instead of full-parameter updates, we optimize LoRA adapters with the same LoRA hyperparameters as in SFT. We set the number of candidate generations per prompt to 4, the sampling temperature to 0.7, the number of training epochs to 3, and the learning rate to 5×10^{-6}.

C.2 Back-translation Audit and Label Noise

We run periodic value-matching audits on sampled instances before they enter $\mathcal{D}_{\text{RL}}$. For each sampled instance, the generated natural-language problem is re-parsed and re-solved, and the instance is discarded when the recovered optimal value deviates from the graph's f_{opt} by more than the proximity

tolerance. On a randomly sampled audit set of 500 instances, 6.2% failed this automatic check and were filtered out. To estimate the residual mismatch that remains after filtering, we manually inspected 100 post-filter instances and identified 2 genuine natural-language/ground-truth semantic mismatches (2%). As with the benchmark protocol in Appendix A.2, this automatic filter checks recovered objective values rather than formulation structure, so residual semantic errors of this kind can remain after filtering.

The staged reward is deliberately designed to limit exposure to label noise. Of the four reward terms, R_{process} (execution) and R_{solvable} (feasibility) are computed by running Gurobi on the generated code and are fully label-independent; only $R_{\text{proximity}}$ and $R_{\text{correctness}}$ depend on f_{opt}. Label noise can therefore corrupt only a bounded portion of the total reward.

C.3 Genetic Algorithm

The weights in fitness function are set to $\beta = 0.3$, $\gamma = 0.3$, and δ follows a linear warm-up schedule from 0 to 0.7 over the first 30% of generations (i.e., δ increases linearly during the warm-up phase and remains at 0.7 thereafter). The GA uses a population size of $\max(30,\ 2N_{\text{target}})$, runs for 10 generations with tournament selection ($k = 3$), crossover rate 0.8, mutation rate 0.5, per-coefficient mutation probability 0.10 with Gaussian noise scale 0.12, size mutation probability 0.20, and elite ratio 15%. Feasibility is verified by Gurobi with a time limit of 300 s and MIP gap 0.01.

Cross-group crossover. Group pairs are selected for crossover only when their compatibility score $s = 0.40\, s_{\text{vtype}} + 0.20\, s_{\text{density}} + 0.25\, s_{\text{size}} + 0.15\, s_{\text{sense}} \geq 0.60$, where the four terms measure cosine similarity of variable-type distributions, constraint-matrix density difference, relative size difference, and optimization-direction consistency, respectively. The operator exchanges contiguous blocks of constraint rows at a random split point $r \in [\lceil n/4 \rceil,\ \lfloor 3n/4 \rfloor]$: rows before r come from one parent and rows after from the other, each carrying its full coefficient vector and bounds. Variable columns are independently inherited per column from a randomly chosen parent, ensuring no orphaned variables. If the offspring is infeasible, a repair step preserves the cross-group constraint matrix A and resets b_l, b_u around a randomly sampled feasible point $\mathbf{x}^*$ to restore feasibility.

C.4 Instance Feasibility and Validity

GenBench-MILP evaluates generated MILP instances by feasibility as well as usefulness Luo et al. [2026]. Usefulness, measured by structural similarity to the 53 seed problem types, is reported in Section 4.3. We report structural validity and solver feasibility here.

Every offspring that enters the candidate pool is structurally valid. The genetic operators preserve validity: crossover keeps exactly one objective node and retains complete variable bounds and metadata, and a deterministic rule-based check and repair step runs before compilation. Every structurally valid candidate graph compiles into executable Gurobi Python code without syntax or solver API errors.

On a random sample of 1,000 compiled models, roughly 86.2% are solved to feasibility or optimality within the 300 s time limit, 10.3% are infeasible with a non-empty irreducible infeasible subset (IIS), and 3.5% encounter solver timeouts. (Approximately 3% of the infeasible samples are retained only for SFT and do not enter weakness analysis or RL.)

C.5 XGBoost

An XGBoost classifier was trained to predict whether the solution outcome for each generated optimization instance was correct. The input representation comprised graph-derived descriptors computed from the optimization model structure, including statistics of the constraint matrix, objective vector, right-hand-side bounds, variable bounds, variable types, and instance-level metadata. In total, 102 features were constructed, as summarized in Table 5. To improve robustness and reduce redundancy, feature selection was performed in three steps. First, features with zero variance were removed. Next, among feature pairs with absolute correlation greater than 0.95, the feature with lower mutual information with the target label was pruned. Finally, the top 20 features ranked by mutual information were retained. Retained features are then standardized by z-score (zero mean, unit variance) using statistics computed on that round's training split. The final XGBoost configuration used 500 boosting rounds, a maximum tree depth of 4, and a learning rate of 0.05. The subsample

ratio and column sampling ratio per tree were set to 0.8 and 0.6, respectively. In addition, the minimum child weight was set to 10, γ was set to 1.0, the L1 regularization coefficient was 1.0, and the L2 regularization coefficient was 5.0. The classifier was trained with the `binary:logistic` objective and evaluated primarily using the AUC metric.

The failure probe is retrained from scratch at every coevolution round on the current model's validation outcomes. Held-out AUC is typically around 0.88 (e.g., 0.8836 for the run in Figure 6). The resulting weakness vector remains a heuristic, correlational prioritization signal rather than a causal account of errors, and we do not report a detailed round-by-round stability analysis of $\mathbf{w}_t$.

Feature Name	Description	Type
A Problem Dimensions		
`n_vars`	Number of decision variables (n_{vars})	Count
`n_cons`	Number of constraints (n_{cons})	Count
`var_con_ratio`	Ratio of variables to constraints ($n_{\text{vars}}/n_{\text{cons}}$)	Ratio
`log_size`	Logarithm of total problem size: $\log(1+n_{\text{vars}}\times n_{\text{cons}})$	Cont.
B Variable Type Composition		
`prop_continuous`	Proportion of continuous (`C`) variables	Ratio
`prop_binary`	Proportion of binary (`B`) variables	Ratio
`prop_integer`	Proportion of general integer (`I`) variables	Ratio
`is_pure_lp`	Binary indicator: problem is a pure LP (no integer or binary variables)	Binary
`has_integer`	Binary indicator: problem contains at least one integer or binary variable	Binary
C Constraint Type		
`prop_equality`	Proportion of equality constraints ($b_l \approx b_u$)	Ratio
D Sparsity and Non-zero Structure		
`sparsity`	Sparsity of constraint matrix $\mathbf{A}$: proportion of zero entries	Ratio
`avg_nnz_per_row`	Mean number of non-zero entries per constraint row	Cont.
`std_nnz_row`	Standard deviation of non-zero counts per row	Cont.
`std_nnz_col`	Standard deviation of non-zero counts per column	Cont.
E Coefficient Statistics (Matrix A)		
`log_max_coeff`	$\log(1+\max\lvert\mathbf{A}_{ij}\rvert)$ for non-zero entries	Cont.
`coeff_cv`	Coefficient of variation of non-zero coefficients in $\mathbf{A}$: $\sigma/(\mu+\epsilon)$	Cont.
`prop_unit_coeff`	Proportion of non-zero entries equal to ± 1	Ratio
`has_frac_coeff`	Binary indicator: at least one fractional (non-integer) coefficient exists in $\mathbf{A}$	Binary
`coeff_range_log`	Log-scale coefficient range in $\mathbf{A}$: $\log(1+\max\lvert a_{ij}\rvert/(\min\lvert a_{ij}\rvert+\epsilon))$	Cont.
F Objective Coefficient Statistics		
`prop_zero_obj`	Proportion of zero entries in objective vector $\mathbf{c}$	Ratio
`log_max_obj`	$\log(1+\max\lvert\mathbf{c}_j\rvert)$ for non-zero objective coefficients	Cont.
`obj_cv`	Coefficient of variation of non-zero objective coefficients	Cont.
G Sign Structure		
`prop_neg_coeff`	Proportion of negative entries in $\mathbf{A}$	Ratio
`prop_mixed_sign_rows`	Proportion of constraint rows with both positive and negative coefficients	Ratio
H Variable Bounds		
`prop_finite_ub`	Proportion of variables with a finite upper bound	Ratio
`avg_bound_width`	Mean width of finite variable bound intervals: $\overline{(u_j - l_j)}$ for bounded variables	Cont.

Continued on next page...

Feature Name	Description	Type
`prop_fixed_vars`	Proportion of variables fixed at a single value ($l_j = u_j$)	Ratio
I Right-Hand Side Statistics		
`log_max_rhs`	$\log(1 + \max \lvert b \rvert)$ over all finite RHS values	Cont.
`rhs_cv`	Coefficient of variation of finite RHS values	Cont.
J Graph-Theoretic Properties		
`graph_density`	Density of the bipartite constraint–variable graph: $\text{nnz}/(n_{\text{cons}} \times n_{\text{vars}})$	Ratio
`degree_std`	Standard deviation of node degrees in the bipartite graph	Cont.
`degree_skewness`	Skewness of the node degree distribution	Cont.
K Objective–Constraint Geometric Similarity		
`obj_cons_max_cos`	Maximum absolute cosine similarity between $\mathbf{c}$ and any constraint row of $\mathbf{A}$	Cont.
`obj_cons_mean_cos`	Mean cosine similarity between $\mathbf{c}$ and constraint rows of $\mathbf{A}$	Cont.
`obj_cons_std_cos`	Standard deviation of cosine similarities between $\mathbf{c}$ and constraint rows	Cont.
L Linear-Algebraic Properties		
`rank_ratio`	Numerical rank ratio of $\mathbf{A}$: $\text{rank}(\mathbf{A})/\min(n_{\text{cons}}, n_{\text{vars}})$	Ratio
`row_norm_cv`	Coefficient of variation of Euclidean row norms of $\mathbf{A}$	Cont.
`prop_tight_cons`	Proportion of constraints with a narrow feasible activity range (tightness ratio < 0.1)	Ratio
`mean_coupling`	Mean number of shared variables between all pairs of constraints (off-diagonal of $\mathbf{A}\mathbf{A}^\top$)	Cont.
M Connectivity		
`n_components`	Number of connected components in the bipartite constraint–variable graph	Count
`is_connected`	Binary indicator: the bipartite graph is connected (single component)	Binary
N NLP Replacement: Problem Scale Proxies		
`graph_size`	Total number of entries in $\mathbf{A}$: $n_{\text{cons}} \times n_{\text{vars}}$	Count
`graph_nnz`	Total number of non-zero entries in $\mathbf{A}$	Count
`graph_n_active_cons`	Number of constraints with at least one non-zero coefficient	Count
`graph_median_coeff`	Median absolute value of non-zero entries in $\mathbf{A}$	Cont.
`graph_mean_nnz_per_col`	Mean number of non-zero entries per variable column	Cont.
`graph_unique_coeff_ratio`	Ratio of distinct non-zero values to total non-zeros in $\mathbf{A}$	Ratio
O NLP Replacement: RHS Numerical Properties		
`graph_finite_rhs_count`	Total count of finite values among all b_l and b_u bounds	Count
`graph_max_finite_abs`	Maximum absolute value across all finite model parameters ($\mathbf{A}$, $\mathbf{c}$, b_l, b_u, lb, ub)	Cont.
`graph_has_frac_rhs`	Binary indicator: at least one fractional RHS value exists	Binary
`graph_has_neg_rhs`	Binary indicator: at least one negative RHS value exists	Binary
`graph_distinct_rhs`	Number of distinct RHS values across all bounds	Count
`graph_rhs_range`	Range of RHS values: $\max(b) - \min(b)$ over all finite bounds	Cont.
P NLP Replacement: Constraint and Variable Type Counts		

Continued on next page...

Feature Name	Description	Type
graph_int_var_count	Number of general integer (I) variables	Count
graph_bin_var_count	Number of binary (B) variables	Count
graph_is_min	Binary indicator: the objective is a minimization	Binary
graph_is_max	Binary indicator: the objective is a maximization	Binary
graph_leq_cons_count	Number of $\leq$ constraints (finite b_u, infinite b_l)	Count
graph_geq_cons_count	Number of $\geq$ constraints (finite b_l, infinite b_u)	Count
graph_eq_cons_count	Number of equality constraints ($b_l \approx b_u$)	Count
graph_zero_row_count	Number of all-zero rows in $\mathbf{A}$ (structurally redundant constraints)	Count
graph_mixed_sign_count	Number of constraints containing both positive and negative coefficients	Count
graph_linking_pair_count	Number of constraints with exactly 2 non-zero entries (binary linking constraints)	Count
graph_frac_coeff_count	Number of fractional (non-integer) coefficients in $\mathbf{A}$	Count
graph_obj_range_ratio	Ratio of maximum to minimum absolute objective coefficient: $\max \lvert c_j \rvert / (\min \lvert c_j \rvert + \epsilon)$	Cont.
graph_unit_row_count	Number of constraint rows whose non-zero entries are all ± 1	Count
graph_range_cons_count	Number of range constraints with both finite and distinct b_l and b_u ($\lvert b_u - b_l \rvert \geq \epsilon$)	Count
graph_obj_density	Proportion of non-zero entries in objective vector $\mathbf{c}$	Ratio
graph_singleton_count	Number of constraints with exactly 1 non-zero entry (single-variable constraints)	Count
graph_dense_row_count	Number of fully-dense rows in $\mathbf{A}$ (all n_{vars} variables appear)	Count
Q NLP Replacement: Constraint Structural Diversity		
graph_cons_diversity	Number of distinct non-zero-count patterns (cardinalities) among constraint rows	Count
R NLP Replacement: Code-Level Structural Proxies		
graph_total_coeff_sum	Sum of absolute values of all entries in $\mathbf{A}$: $\sum_{i,j} \lvert a_{ij} \rvert$	Cont.
graph_n_rows	Number of constraint rows in the model	Count
graph_n_cols	Number of variable columns in the model	Count
graph_bounded_cons_count	Number of constraints bounded on at least one side (finite b_l or b_u)	Count
graph_has_identical_rows	Binary indicator: at least two rows of $\mathbf{A}$ share an identical sign pattern	Binary
graph_max_col_nnz	Maximum number of non-zero entries in any single variable column	Count
graph_has_bigM	Binary indicator: Big-M pattern detected (a coefficient exceeds $50\times$ the median non-zero value)	Binary
graph_vtype_diversity	Number of distinct variable types present (C, I, B)	Count
graph_obj_nnz_count	Number of non-zero entries in objective vector $\mathbf{c}$	Count
graph_all_finite_ub	Binary indicator: all variables have finite upper bounds	Binary
graph_var_group_count	Number of distinct variable subsets (support sets) appearing across constraint rows	Count
graph_block_mixing_ratio	Proportion of constraints whose non-zeros span both the left and right halves of the variable index set	Ratio
graph_n_var_names	Number of named decision variables in the model	Count
graph_n_con_names	Number of named constraints in the model	Count
graph_avg_row_abs_sum	Mean sum of absolute coefficients per constraint row: $\overline{\sum_j \lvert a_{ij} \rvert}$	Cont.
graph_max_var_participation	Maximum number of constraints in which any single variable appears	Count
graph_total_distinct_values	Total number of distinct numeric values across $\mathbf{A}$, $\mathbf{c}$, b_l, b_u, lb, ub	Count

Continued on next page...

Feature Name	Description	Type
S NLP Replacement: Internal Consistency Ratios		
`graph_rhs_to_cons_ratio`	Ratio of finite RHS count to number of constraints	Ratio
`graph_rhs_coeff_mag_ratio`	Ratio of mean absolute RHS to mean absolute coefficient in $\mathbf{A}$	Cont.
`graph_nnz_to_cons_ratio`	Ratio of total non-zeros in $\mathbf{A}$ to number of constraints	Cont.
`graph_active_cons_ratio`	Proportion of constraints with at least one non-zero coefficient	Ratio
`graph_obj_A_value_overlap`	Fraction of distinct objective coefficient values that also appear as entries in $\mathbf{A}$	Ratio
`graph_A_only_value_count`	Number of distinct numeric values appearing in $\mathbf{A}$ but not in $\mathbf{c}$	Count
`graph_c_only_value_count`	Number of distinct numeric values appearing in $\mathbf{c}$ but not in $\mathbf{A}$	Count
`graph_col_jaccard`	Mean Jaccard similarity of constraint participation sets over sampled pairs of variable columns	Ratio
`graph_complexity_score`	Composite complexity score: $n_{\text{cons}} \times$ diversity $+$ finite RHS count	Cont.
`graph_structural_score`	Structural complexity score: bounded constraints $+ n_{\text{vars}} + 2 \cdot \mathbf{1}[\text{mixed}] +$ group count	Cont.
`graph_complexity_diff`	Difference between composite and structural complexity: `graph_complexity_score` $-$ `graph_structural_score`	Cont.
T Instance Metadata		
`problem_type_enc`	Label-encoded integer representing the problem category (e.g. LP, MIP, MIQP)	Int.
`inst_n_vars`	Number of variables parsed from the instance identifier	Count
`inst_n_cons`	Number of constraints parsed from the instance identifier	Count

C.6 Prompt Templates

C.6.1 Baseline Prompt

Baseline Prompt Template for optimization modeling

```
Below is an operations research question. Build a mathematical model and
corresponding python code using `gurobipy` that appropriately addresses the
question.

# Question:
{question}

# Response:
```

C.6.2 Reverse Data Generate Prompt

Prompt for Cross-Group Example Generation

```
As an Operations Research Expert, your task is to create a SINGLE coherent
optimization problem description that naturally combines two problem types
into one unified scenario, with a CLEAR emphasis on one type as the primary
framework.

## Primary Problem Type (DOMINANT): <DOM_SUBCLASS>

Mathematical Model:
<DOM_FORMULA>

Example:
<DOM_EXAMPLE>

## Secondary Problem Type (supplementary): <MIN_SUBCLASS>

Mathematical Model:
<MIN_FORMULA>

Example:
<MIN_EXAMPLE>

## Your Task:
Create ONE natural language description of an optimization problem that:
1. Presents a single, realistic scenario (e.g., a company, a logistics
operation, a planning task)
2. Uses <DOM_SUBCLASS> as the PRIMARY framework and core storyline
3. Naturally weaves in <MIN_SUBCLASS> elements as SECONDARY considerations
4. The <DOM_SUBCLASS> aspects should be clearly dominant (roughly 60-70% of
the problem), while <MIN_SUBCLASS> provides the remaining enrichment
5. Includes specific numerical parameters (costs, capacities, demands, etc.)
to make it concrete
6. Clearly states what needs to be optimized
7. Describes all constraints naturally within the narrative
8. Uses appropriate domain terminology from both fields
9. Does NOT contain any mathematical formulas or variable names

For example, if the dominant type is "Knapsack" and secondary is "Bin
Packing", you might describe a purchasing manager who primarily needs to
select which products to buy within a budget, and additionally must arrange
selected products into shipping containers.

## Required Output:
Provide ONLY the natural language problem description. No headers, no
meta-commentary, no formulas. Just the problem scenario.
```

Prompt for Single-Group Reverse Generation

```
As an Operations Research Expert, analyze the given LP data and generate a
natural language description.

Input LP Data:
<LP_DATA>

Mathematical Model Reference:
<FORMULA>

Application Domain: <SUBCLASS>
```

```
Reference Example:
<EXAMPLE>

## Required Output:
Generate a clear, detailed natural language description of this optimization
problem that:
- Describes a concrete, realistic scenario in the <SUBCLASS> domain
- States all decision variables and what they represent
- Specifies the objective (minimize/maximize) and what it means in the
scenario
- Incorporates ALL constraints naturally within the narrative
- Includes ALL numerical parameters (coefficients, bounds, right-hand sides)
exactly as they appear in the LP data
- Uses appropriate domain terminology
- Does NOT include any mathematical formulas, variable names like x0/x1, or
LP file content

CRITICAL: Every numerical value in the LP data must appear somewhere in your
description. Do not omit or approximate any parameter.

Provide ONLY the natural language description, nothing else.
```

Prompt for Dominant Cross-Group Generation

```
As an Operations Research Expert, analyze the given LP data and generate a
natural language description.

This optimization problem primarily belongs to "<DOM_SUBCLASS>" (approx.
<DOM_RATIO>\%), with additional constraints from "<MIN_SUBCLASS>" (approx.
<MIN_RATIO>\%).

Input LP Data:
<LP_DATA>

Primary Mathematical Model Reference (<DOM_SUBCLASS>):
<DOM_FORMULA>

Cross-Domain Reference Example (showing how <DOM_SUBCLASS> and <MIN_SUBCLASS>
can be combined in one scenario):
<CROSS_EXAMPLE>

## Required Output:
Following the style of the cross-domain reference example above, generate a
natural language description that:
- Frames the problem primarily as a <DOM_SUBCLASS> problem
- Naturally incorporates the <MIN_SUBCLASS>-style constraints as part of the
same scenario
- Describes a concrete, realistic scenario
- States all decision variables and what they represent
- Specifies the objective (minimize/maximize) clearly
- Incorporates ALL constraints naturally within the narrative
- Includes ALL numerical parameters (coefficients, bounds, right-hand sides)
EXACTLY as they appear in the LP data
- Does NOT include any mathematical formulas, variable names like x0/x1, or
LP file content

CRITICAL: Every numerical value in the LP data must appear somewhere in your
description. Do not omit or approximate any parameter. The description must
be fully self-contained - a reader should be able to reconstruct the LP
formulation from your description alone.
```

```
Provide ONLY the natural language description, nothing else.
```

Prompt for Mixed Cross-Group Generation

```
As an Operations Research Expert, analyze the given LP data and generate a
natural language description.

This optimization problem combines two domains:
- Primary: "<DOM_SUBCLASS>" (approx. <DOM_RATIO>\%)
- Secondary: "<MIN_SUBCLASS>" (approx. <MIN_RATIO>\%)

Input LP Data:
<LP_DATA>

Primary Mathematical Model Reference (<DOM_SUBCLASS>):
<DOM_FORMULA>

Cross-Domain Reference Example (showing how to combine <DOM_SUBCLASS> and
<MIN_SUBCLASS> naturally):
<CROSS_EXAMPLE>

## Required Output:
Using the cross-domain reference example as a style guide, generate a natural
language description that:
- Creates a unified, realistic scenario combining aspects of both
<DOM_SUBCLASS> and <MIN_SUBCLASS>
- The primary framing should be from the <DOM_SUBCLASS> perspective
- <MIN_SUBCLASS> constraints should be woven in naturally
- States all decision variables and what they represent
- Specifies the objective clearly
- Incorporates ALL constraints within the narrative
- Includes ALL numerical parameters EXACTLY as in the LP data
- Does NOT include any mathematical formulas, variable names like x0/x1, or
LP file content

CRITICAL: Every numerical value in the LP data must appear somewhere in your
description. Do not omit or approximate any parameter. The description must
be fully self-contained - a reader should be able to reconstruct the LP
formulation from your description alone.

Provide ONLY the natural language description, nothing else.
```

Prompt for Fusion Cross-Group Generation

```
As an Operations Research Expert, analyze the given LP data and generate a
natural language description.

This optimization problem equally combines:
- "<DOM_SUBCLASS>" (approx. <DOM_RATIO>\%)
- "<MIN_SUBCLASS>" (approx. <MIN_RATIO>\%)

Input LP Data:
<LP_DATA>

Cross-Domain Reference Example (demonstrating how <DOM_SUBCLASS> and
<MIN_SUBCLASS> merge into one scenario):
<CROSS_EXAMPLE>
```

```
## Required Output:
Using the cross-domain reference example as a style guide, generate a natural
language description that:
- Creates a single coherent, realistic scenario integrating BOTH
<DOM_SUBCLASS> and <MIN_SUBCLASS> equally
- Both problem aspects should be equally prominent in the narrative
- States all decision variables and what they represent
- Specifies the objective clearly
- Incorporates ALL constraints within the narrative
- Includes ALL numerical parameters EXACTLY as in the LP data
- Does NOT include any mathematical formulas, variable names like x0/x1, or
LP file content

CRITICAL: Every numerical value in the LP data must appear somewhere in your
description. Do not omit or approximate any parameter. The description must
be fully self-contained - a reader should be able to reconstruct the LP
formulation from your description alone.

Provide ONLY the natural language description, nothing else.
```

Prompt for Self-Critique

```
As an Operations Research Expert, rigorously verify if the generated problem
description accurately and completely represents the LP data.

Input LP Data:
<LP_DATA>

Generated Problem Description:
<DESCRIPTION>

## Verification Checklist:
1. Count decision variables in LP data vs description - do they match?
2. For EACH variable: are bounds (lower, upper) correctly stated?
3. For EACH variable: is the type (integer, binary, continuous) correctly
described?
4. Is the objective function direction (min/max) correct?
5. For EACH objective coefficient: does the description include the exact
value?
6. Count constraints in LP data vs description - do they match?
7. For EACH constraint: are ALL coefficients exactly correct?
8. For EACH constraint: are the right-hand side values (bounds) exactly
correct?
9. Are there any numerical values in the LP data that are missing from the
description?
10. Are there any numerical values in the description that do NOT appear in
the LP data?

## Required Output:
If ALL checks pass perfectly:
"Complete Instance"

If ANY discrepancy exists:
"Incomplete Instance:
[List EVERY specific discrepancy with exact values from LP data that are
wrong or missing]"
```

Table 6: Graph-based evolution versus text-only evolution under the same SFT warm-up, RL recipe, and per-round data budget. Pass@1 accuracy (%) on six OR benchmarks.

Setting	NL4OPT	MAMO		NLP4LP	CompOR	IndOR	Avg.
		EasyLP	ComplexLP				
Text-only (MILP-Evolve)	88.3	96.1	50.5	91.0	44.4	47.6	69.7
EvoOptiGraph	96.7	98.0	69.4	97.2	50.0	57.1	78.1

Prompt for Self-Refinement

```
As an Operations Research Expert, fix the problem description based on the
criticism.

Criticism:
<CRITICISM>

If the criticism says "Complete Instance", output exactly: "Nothing need to
refine"

Otherwise, fix ALL identified issues:

LP Data (ground truth):
<LP_DATA>

Scenario Context: <SCENARIO_CONTEXT>

Current Description (contains errors):
<INITIAL_DESCRIPTION>

## Required Output:
If criticism is "Complete Instance":
Output "Nothing need to refine"

Otherwise, generate a CORRECTED description that:
- Fixes every single discrepancy identified in the criticism
- Maintains the same narrative style and scenario
- Includes ALL numerical parameters EXACTLY as in the LP data
- Does NOT include any mathematical formulas or LP file content
- Is a complete, standalone description (not a patch or diff)

Output ONLY the corrected natural language description, nothing else.
```

D Comparison with Text-Only Evolution

We compare graph-based evolution with a text-only evolution baseline based on MILP-Evolve [Li et al., 2024], an LLM-based evolutionary framework that generates diverse MILP classes and an unlimited number of instances. In this baseline, an LLM mutates the natural-language description or the solver code directly. EvoOptiGraph edits the bipartite constraint graph and uses the weakness vector to guide which instances are generated. The SFT warm-up, RL recipe, and per-round data budget are otherwise identical. Text-only evolution does not use the weakness vector.

Table 6 reports Pass@1 under this matched setting. Text-only evolution is lower on every benchmark, and the macro-average gap is 8.4 percentage points (78.1 vs. 69.7). Two factors account for the difference. First, because text-only evolution has no weakness vector, new instances are not steered toward the model's current failure modes, so the training data are less targeted. Second, text-level mutations often produce semantically ambiguous constraints or invalid formulations that fail Gurobi compilation. The required re-sampling lowers the effective data yield and reduces the diversity of the instances that remain.

E Supplementary Technical Details

E.1 Formal Definition of Graph Edit Distance (GED) and Its Application in This Paper

Graph Edit Distance (GED) is a classic measure of structural dissimilarity between two graphs [Gao et al., 2010, Gasse et al., 2019, Fan et al., 2023]. For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, GED is defined as the minimum cost of a sequence of edit operations (insertion, deletion, and substitution of nodes and edges) that transforms G_1 into G_2. Its mathematical definition is:

$$\mathrm{GED}(G_1, G_2) = \min_{(o_1,\ldots,o_k)\in\mathcal{P}(G_1,G_2)} \sum_{i=1}^{k} c(o_i),$$

where $c(o_i)$ is the cost of operation o_i, and $\mathcal{P}(G_1, G_2)$ is the set of all valid sequences of operations.

In this paper, we adopt unit costs (all operations have cost 1) and compute GED efficiently using a bipartite matching approximation algorithm [Blumenthal et al., 2020]. In Section 3.2, GED is used to compute the diversity term $\tilde{f}_{\text{div}}(\mathcal{G})$, i.e., the normalized average GED between the current individual and other individuals in the population. A larger GED indicates a more significant structural difference, thereby encouraging the evolution to explore novel graph structures. This design ensures structural diversity of the population, providing a foundation for subsequent difficulty selection and weakness guidance.

E.2 How the Weakness Signal Guides Generation

The weakness signal guides EvoGenerate through the weakness-alignment term of the fitness function. In each coevolution round, the current model is evaluated on the held-out validation set to obtain its error set $\mathcal{E}_t$. An XGBoost classifier is trained to predict correctness from the structural feature vector $\phi(\mathcal{G})$, and SHAP values on the error cases form the weakness vector

$$w_k = \frac{1}{|\mathcal{E}_t|} \sum_{i\in\mathcal{E}_t} \mathrm{SHAP}_{i,k},$$

where each coordinate is the direction along which a structural feature is most associated with failure.

This vector enters generation through the fitness

$$f(\mathcal{G}) = \beta\tilde{f}_{\text{diff}} + \gamma\tilde{f}_{\text{div}} + \delta\tilde{f}_{\text{weak}}, \qquad \tilde{f}_{\text{weak}} = \cos\big(\phi(\mathcal{G}), \mathbf{w}_t\big).$$

The weight δ follows the schedule in Section C.3: it increases linearly from 0 to 0.7 over the first 30% of generations and stays at 0.7 thereafter.

A candidate whose structure is closer to the model's current failure regime receives a higher fitness, so tournament selection is more likely to keep and propagate it. The resulting dataset $\mathcal{D}_{\text{RL}}$ is therefore biased toward the model's structural blind spots. These instances drive the GRPO update. The next round re-diagnoses the updated model and produces $\mathbf{w}_{t+1}$, so the training distribution co-evolves with the model.

This construction remains heuristic. Cosine alignment with a SHAP-aggregated vector is a convenient targeting mechanism rather than a guarantee that the selected instances cause the observed failures. The probe itself is specified in Appendix C.5.

E.3 t-SNE Dimensionality Reduction and Its Application in Weakness Analysis

Brief Introduction to t-SNE t-SNE is a nonlinear dimensionality reduction technique that preserves local structure by minimizing the KL divergence between probability distributions in the high-dimensional and low-dimensional spaces [Van der Maaten and Hinton, 2008]. It defines high-dimensional similarities p_{ij} and low-dimensional similarities q_{ij} and optimizes the low-dimensional coordinates via gradient descent.

Application in Weakness Analysis We use t-SNE to visualize the feature vectors $\phi(\mathcal{G})$ to validate the effectiveness of the extracted features. The procedure is as follows:

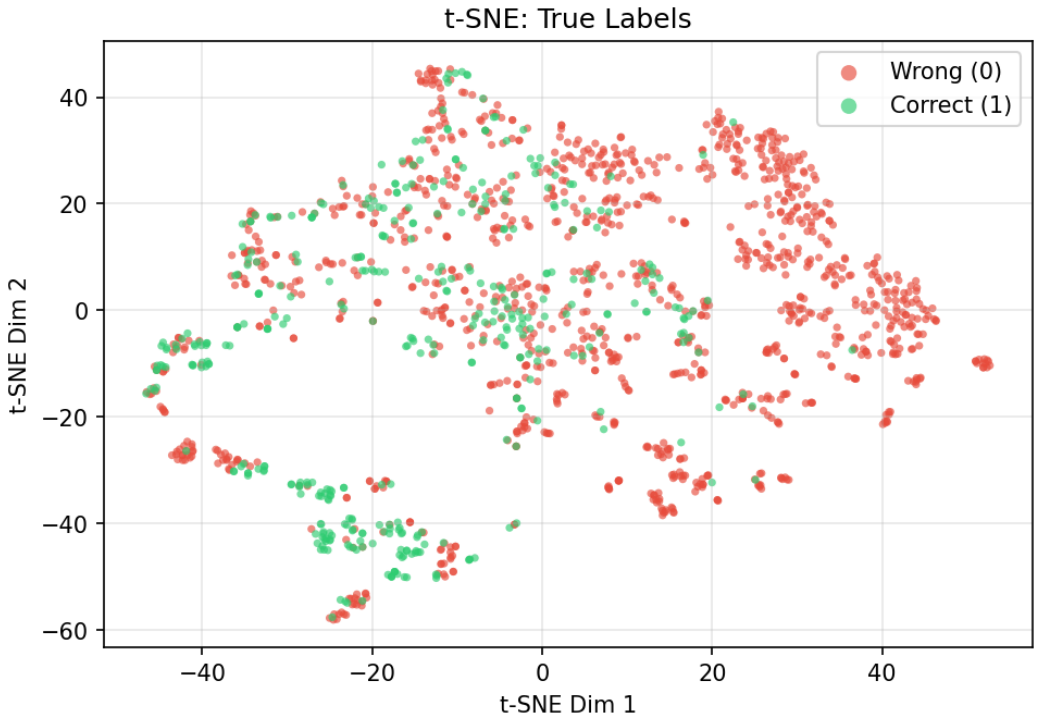

Figure 6: An Example of t-SNE Feature Visualization

1. Select correctly and incorrectly solved instances from the validation set V_{val}.
2. Extract the d-dimensional feature vector $\phi(\mathcal{G})$ for each instance (see Section 3.3 for details).
3. Apply t-SNE to project the features into two dimensions for visualization.

In the resulting plot, correctly and incorrectly solved instances are marked with different colors. Figure 6 shows one representative example from a particular run. If the two classes exhibit visible clustering patterns, this provides visual evidence that the feature vector $\phi(\mathcal{G})$ captures structural information relevant to model errors. For the example shown in Figure 6, the corresponding classifier achieves an AUC of 0.8836.

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: We have made main claims that accurately reflect the paper's contributions and scope in abstract and introduction.

   Guidelines:

   - The answer [N/A] 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 [N/A] 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 paper discuss the limitations of this work.

   Guidelines:

   - The answer [N/A] 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: [N/A]

   Justification: This paper does not include the content of the theoretical analysis results that require assumptions or proof.

   Guidelines:

   - The answer [N/A] 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: The paper provides all the information needed to reproduce the main experimental results.

Guidelines:

- The answer [N/A] 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: [No]

Justification: We do not release code with this submission. The evaluation benchmarks and the Qwen3-8B backbone are public, and the algorithm, prompts, and training configuration are described in the paper and appendix.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.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://neurips.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) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specify the all details of experiment, including hyper-parameters, the rationale for their selection and underlying model.

   Guidelines:

   - The answer [N/A] 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: Appendix A.3 reports EvoOptiGraph Pass@1 as mean $\pm$ standard deviation (Table 4). Each entry is the mean over 3 evaluation runs and 2 training seeds. Baseline numbers remain published point estimates.

   Guidelines:

   - The answer [N/A] 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: The paper provides sufficient details about the compute resources used in the experiments, including the type of hardware (e.g.,GPU). It also reports resource requirements for the main experimental runs, which would help readers reproduce the results.

Guidelines:

- The answer [N/A] 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: The research conducted in the paper conform with the NeurIPS Code of Ethics in every respect.

Guidelines:

- The answer [N/A] 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: [N/A]

Justification: The work performed has no direct societal impact, as it relies solely on calling LLMs and their behavior is fully controlled. The approach does not involve any external factors or unintended consequences that could potentially affect society.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] 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., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] 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: The creators of the data and LLMs employed in this research have appropriately acknowledged and adhered to the respective licenses governing their use for research purposes.

    Guidelines:

    - The answer [N/A] 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: [N/A]

    Justification: The paper does not release the code.

    Guidelines:

- The answer [N/A] 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: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] 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: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] 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 rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLM was used during synthetic data construction via back-translation. In addition, we performed supervised fine-tuning (SFT) and reinforcement learning (RL) on the Qwen3-8B model for the intelligent modeling task. However, these usages serve as implementation and experimental components rather than the primary technical novelty of the proposed method.

Guidelines:

- The answer [N/A] 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 in the NeurIPS handbook for what should or should not be described.