AutoHOpt: From Natural Language to Model-Conditioned Metaheuristic Search

Anonymous authors
Paper under double-blind review

Abstract

Existing language-to-optimization systems primarily treat mathematical models as inputs to solvers. Yet the same constraint structure that defines feasibility can also guide how solutions are constructed and improved. We introduce AutoHOpt, a framework that turns natural-language requirements into model-conditioned metaheuristic search. AutoHOpt first recovers a source-reviewed executable model, OptSpec, and then uses its decisions and constraints to generate SearchIR, a task-specific search program. The accepted model remains fixed while execution feedback refines initialization and neighborhood-repair operators, which run within adaptive large neighborhood search without per-iteration LLM calls. Across ten external benchmarks, AutoHOpt achieves the highest mean zero-gap success on eight. On IndustryOR and MIPLIB-NL, it reaches 83.5% and 78.8%, exceeding the strongest baseline by 20.2 and 9.5 percentage points, respectively. Ablations and common-budget controls demonstrate the contributions of modeling feedback and model-conditioned search. These results establish executable models as an effective interface between natural-language problem understanding and automated search design.

1 Introduction

Industrial operations depend on combinatorial decisions: factories sequence jobs on shared machines, logistics networks assign demand to facilities, and autonomous fleets allocate tasks and plan routes. Many fundamental versions of these problems are NP-hard (Garey et al., 1976; Lenstra & Rinnooy Kan, 1981). The difficulty starts before search: operational requirements must be translated into objectives, constraints and decision variables. Changes in production processes or resource policies can require revising both the model and its solution procedure.

A widely used approach formulates a mixed-integer linear program (MILP) and invokes a general-purpose solver. Language-to-optimization systems reduce the modeling effort by generating formulations and solver programs from natural-language descriptions (AhmadiTeshnizi et al., 2024; Xiao et al., 2024; Huang et al., 2025a). Execution-aware systems such as NEMO strengthen this approach through automatic coding, validation and repair (Song et al., 2026). The resulting model primarily specifies the problem for an optimization backend, whose search procedures are provided separately.

Industrial requirements, however, often combine tightly coupled operational rules. A scheduling task may involve alternative machines, maintenance windows, shared technicians and delivery deadlines. Encoding these relationships requires careful choices of auxiliary variables and linearizations, with search performance sensitive to formulation strength (Vielma, 2015). The relationships also contain useful algorithmic information: they identify which decisions interact and how a partially modified solution can be reconstructed.

We use this structure to connect problem modeling with search design. An executable model supplies the decisions, objective and feasibility checks needed by a metaheuristic; its constraints can additionally guide the construction and improvement of candidate solutions. This creates a route from natural-language requirements to a task-specific search procedure, without requiring a MILP formulation as an intermediate step.

AutoHOpt realizes this route through OptSpec, a source-reviewed executable model, and SearchIR, the search program generated from that model. Figure 1 summarizes the workflow;

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

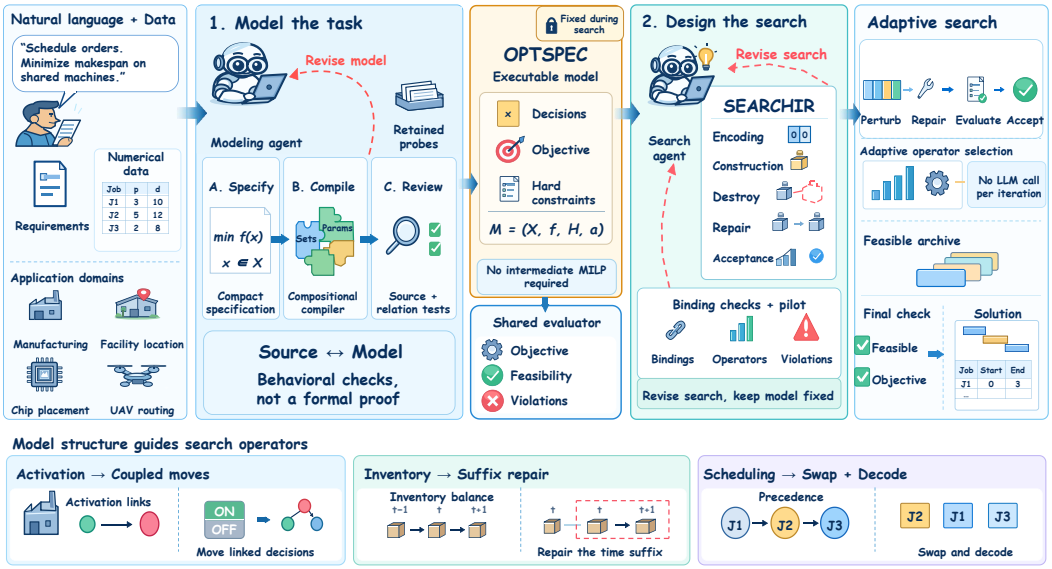


Figure 1: Overview of AutoHOpt. Source-reviewed modeling produces executable OPTSPEC; its semantics remain fixed while the search agent generates and tests SEARCHIR operators. The shared evaluator guides adaptive search, feasible-solution retention and final checking, with no LLM call per iteration. Lower panels show how activation, inventory and scheduling structure guide coupled moves, suffix repair and swap-and-decode operators.

Section 4 details how model structure guides operator generation and execution. Our contributions are threefold:

- **Natural language to model-conditioned metaheuristics.** We introduce AutoHOpt, an end-to-end framework that transforms natural-language OR problems into task-specific metaheuristic search. Its OPTSPEC–SEARCHIR interface uses recovered objectives and constraints to generate and validate construction and neighborhood-repair programs. This connects problem understanding to algorithm design without requiring an intermediate MILP formulation or an external solver.
- **State-of-the-art results on challenging OR benchmarks.** AutoHOpt achieves the highest mean zero-gap success on eight of ten external benchmarks. On IndustryOR and MIPLIB-NL, it reaches 83.5% and 78.8%, improving over NEMO by 20.2 and 9.5 percentage points, respectively.
- **A large-scale benchmark for industrial optimization.** We introduce COPBENCH, comprising 1,440 tasks across manufacturing scheduling, facility location, chip placement and UAV planning, with four problem scales. The benchmark provides a testbed for evaluating methods on larger and more complex industrial problems. AutoHOpt leads the evaluated methods in both full-suite feasibility (87.4%) and solution quality, halving NEMO's mean objective gap from 4.8% to 2.4% on the common feasible intersection across all methods.

2 RELATED WORK

Language-to-optimization modeling. OptiMUS and Chain-of-Experts generate and refine formulations and solver programs; ORLM and LLMOPT specialize language models for optimization tasks (AhmadiTeshnizi et al., 2024; Xiao et al., 2024; Huang et al., 2025a; Jiang et al., 2025). NEMO coordinates executable optimizer and simulator implementations with feedback and experience reuse (Song et al., 2026). These approaches are solver-oriented, although their supported problem classes extend beyond MILP. AutoHOpt instead uses the recovered objective and constraints as the interface for constructing a metaheuristic. NL4Opt, MAMO and MIPLIB-NL provide complementary tests of linear word problems, mixed modeling structures and industrial MILPs (Ramamonjison et al., 2022; Huang et al., 2025b; Li et al., 2026).

Verification and structured representations. Opt-Verifier, VeriSimpl and IR2Solve examine correspondence, diagnostic queries and typed model compilation (Liu et al., 2026; Abdul Rahman et al., 2026; Zhu et al., 2026). Self-Refine and Reflexion use iterative feedback to revise generated artifacts (Madaan et al., 2023; Shinn et al., 2023). Our modeling loop turns source relationships into executable probes and retains failed scenarios across revisions. A second feedback loop operates on SEARCHIR: it tests whether the proposed algorithm manipulates the accepted model's decisions correctly and constructs feasible candidates. The two loops address distinct error sources without conflating a search failure with a change in the problem definition. Both remain internal to model construction and search design; neither replaces an independent assessment of the returned decisions against the source requirements.

Automated heuristic design and ALNS. Hyper-heuristics and algorithm selection organize reusable procedures (Burke et al., 2013; Kerschke et al., 2019). FunSearch, EoH, ReEvo and LLaMEA generate or improve heuristic code using numerical feedback (Romera-Paredes et al., 2024; Liu et al., 2024; Ye et al., 2024; van Stein & Bäck, 2025). EoH evolves natural-language ideas together with executable heuristics; ReEvo combines heuristic evolution with short- and long-term reflection. Their optimization settings provide a heuristic interface and an evaluation procedure. AutoHOpt connects this line of work to natural-language modeling. Rather than assuming that the optimization task and decision encoding are already available, it recovers them in OPTSPEC and uses their exact structure to generate executable construction and neighborhood-repair programs. This interface also makes the compatibility between model and algorithm inspectable. cuGenOpt also connects an LLM modeling assistant to a GPU metaheuristic engine (Liu, 2026). AutoHOpt generates and validates task-specific operators from a source-reviewed executable model whose semantics remain fixed during search revision. Appendix J.1 compares direct task-to-code generation and adaptations of ReEvo and EoH on the same frozen models.

3 PROBLEM SETTING

Given a natural-language task d, numerical data a and budget B, the system constructs a model and returns a candidate solution. We write

$$M = (\mathcal{X}, f, \mathcal{H}, a), \qquad \min_{x \in \mathcal{X}} f(x; a) \quad \text{s.t.} \quad h_k(x; a) = \text{true} \quad \forall h_k \in \mathcal{H}. \tag{1}$$

Here $\mathcal{X}$ specifies decision domains, f computes the objective, and $\mathcal{H}$ contains executable constraints. Maximization uses the opposite objective orientation during ranking. Supported decisions include bounded numeric variables and combinatorial assignments, subsets and permutations. Logical, temporal and nonlinear relations can be evaluated directly when supported by the compiler; they need not first be linearized.

OPTSPEC is the executable interpretation of M. SEARCHIR describes a search procedure A_M conditioned on that interpretation. This separates two responsibilities: deciding which candidates are valid, and deciding which candidates to explore. The solution is checked against the same OPTSPEC after search. Source-level evaluation additionally asks whether the interpretation faithfully captures the original task. In particular, let $F_{\text{src}}(x; d, a)$ denote source-level feasibility and $F_M(x)$ model feasibility. The implication $F_M(x) = 1 \Rightarrow F_{\text{src}}(x; d, a) = 1$ requires a faithful model and cannot be established by reusing F_M. An omitted business rule can therefore pass every model-based runtime check. We use "model-feasible" for the runtime guarantee and reserve "source-validated" for an additional check of the returned decision against the original task. Appendix K describes source validation on a recorded execution.

4 AutoHOpt

Figure 1 shows how AutoHOpt connects source-reviewed modeling to executable search design through two representations:

$$d \longrightarrow \text{OPTSPEC} \longrightarrow \text{SEARCHIR} \longrightarrow \widehat{x}. \tag{2}$$

The modeling stage builds OPTSPEC, which captures the decision domains, objective and hard constraints. It separates fixed source facts from revisable interpretations and uses executable feedback to

correct specific relationships. Once accepted, this model supplies the decision coordinates and evaluator for SEARCHIR. The search agent generates initialization and coupled destroy/repair functions, with local improvement when useful. Modeling failures revise the interpretation; search failures revise the algorithm while preserving the accepted model.

The lower panels of Figure 1 illustrate how mathematical structure becomes search behavior. Activation relationships motivate coupled moves, inventory balances motivate suffix repair, and scheduling precedence motivates swaps followed by decoding. The runtime executes these operators within adaptive large neighborhood search (ALNS) (Ropke & Pisinger, 2006), supplies adaptation and acceptance, and evaluates each candidate against the shared model. A feasible archive retains solutions for final verification without an LLM call at each iteration. This division lets generated programs exploit task-specific structure while the host maintains consistent evaluation and search control.

4.1 FROM NATURAL LANGUAGE TO EXECUTABLE OPTSPEC

The modeling input contains the complete task description and numerical data. The LLM produces a compact declaration of indexed sets, parameters, decision domains, objective terms, and constraint relationships. It can additionally declare derived quantities and reusable modeling components. A compiler expands these declarations into expressions, assigns identifiers, resolves indexing, and constructs the evaluator. This division concentrates generation on mathematical choices while moving repetitive bookkeeping into deterministic code. Appendix E specifies the model interface and runtime checks.

Source text, tables and literal numbers remain fixed during revision; variable meanings, units and derived relationships belong to the candidate interpretation. Literal grounding, dimensional checks and source review separately verify provenance, composition and meaning. Appendix E illustrates how a unit-conversion error is corrected without changing the source data.

Recurring industrial relationships are compiled from composable components. For inventory, the LLM specifies the order of receipt and consumption. The compiler expands balances and the appropriate capacity checks:

$$I_t = I_{t-1} + u_t - v_t, \qquad \begin{cases} I_t \le C, & \text{capacity after consumption,} \\ I_{t-1} + u_t \le C, & \text{capacity after receipt.} \end{cases} \tag{3}$$

The two checks encode different operational requirements even though their end-of-period balances coincide. Other components cover integer batches, activation–production links, complete shift-coverage intervals, and marginal versus all-units prices. The LLM selects and configures these structures from the description; the compiler handles their expansion. Components can therefore be combined across problem families rather than tied to benchmark identities.

The accepted representation defines $M = (\mathcal{X}, f, \mathcal{H}, a)$ and exposes

$$E_M(x) = \big(f(x;a), F_M(x), \{v_k(x;a)\}_{k \in K(M)}\big), \qquad F_M(x) = \bigwedge_{h \in \mathcal{H}} h(x;a), \tag{4}$$

where v_k identifies a violated relationship. Integer, continuous, binary, and combinatorial decisions retain their declared domains. Execution is chosen after these domains and expressions are known. Source bounds are recorded separately from bounds justified by capacities or supported objective-level-set arguments; an arbitrary exploration range cannot become a new source constraint.

4.2 LEARNING FROM FAILURES AND REVIEWING SUCCESSFUL MODELS

A successful compilation establishes that a model can execute. We next examine whether its behavior matches the description. The review role reads both objects and proposes small distinguishing scenarios: the cost of buying one batch, the inventory immediately after replenishment, the periods covered by one shift, or the behavior on either side of a price breakpoint. Scenarios include legal and illegal decisions, so the review can expose both missing restrictions and unjustified exclusions. These checks connect source-to-model coverage with model-to-source justification through observable outcomes.

Each scenario is an executable probe $b = (x_b, t_b, y_b)$: a decision assignment, an evaluation target, and its expected result derived from the source. Targets include the objective, an individual relation,

a derived quantity, integrality, and complete feasibility. The host executes the probe and compares $E_{M,t_b}(x_b)$ with y_b. Thus a reviewer must supply a testable discrepancy rather than an unsupported approval judgment.

Failures return to the modeler as a structured record containing the failing stage, affected expression or field, diagnostic message, and previous revision. Source counterexamples are retained across attempts:

$$\mathcal{B}_{r+1} = \mathcal{B}_r \cup \mathcal{B}_r^{\text{new}}. \tag{5}$$

Every revised candidate is compiled and checked against this bank before a new review. When decision names or representations change, probes can be explicitly migrated while preserving their source-derived expected outcomes. Local edits preserve unrelated constraints; a broader change of interpretation requires a source-based explanation and renewed checking. Repeated occurrences of the same failure trigger a request to reconsider the interpretation or representation, so later attempts are explicitly directed toward a different explanation.

This loop turns unsuccessful attempts into reusable evidence and makes successful attempts pass a behavioral review. The resulting status records the checks that were satisfied. Independent evaluation against the original task remains necessary because a finite set of probes cannot establish complete semantic equivalence for unrestricted natural language.

4.3 Generating model-conditioned search programs in SearchIR

Once accepted, the model becomes a fixed contract for search design. The compiler provides a projection Π from mathematical decisions to runtime coordinates: an indexed scalar maps to its exact serialized key, an assignment retains its item and category labels, and a permutation is represented as a whole sequence. The designer receives this projection together with resolved data, variable domains, objective expressions, and constraint structure. Consequently, search operators manipulate the same decisions that the model evaluator checks.

We write the search representation as

$$S = (h_M, \rho, I, \mathcal{D}, \mathcal{R}, \mathcal{L}, A, \theta, B), \tag{6}$$

where h_M binds the model context, ρ is the decision encoding, I is the constructor, $\mathcal{D}$ and $\mathcal{R}$ are perturbation and reconstruction portfolios, $\mathcal{L}$ denotes local improvement, A is the acceptance rule, and (θ, B) contain controller settings and budgets. The runtime supplies the adaptive controller and acceptance mechanism. The LLM designs the neighborhood structures and emits initialization, destroy and repair source code specialized to the accepted model. Local improvement is implemented within the generated repair functions when useful.

This is where mathematical structure becomes algorithmic structure. In a fixed-charge model, changing a facility's open/closed decision should be coupled with the quantities assigned to it. For inventory, changing a receipt may require reconstructing a suffix of the replenishment plan. For scheduling, insertion or exchange moves can be followed by a decoder that respects readiness, eligibility, and shared resources. The designer uses the active relationships to identify such coupled moves rather than perturbing each coordinate independently.

A destroy function returns a copied state and an explicit set J of released coordinates. Every repair returns a complete candidate satisfying the interface contract

$$x'_i = x_i \qquad \text{for every } i \notin J. \tag{7}$$

All decisions that must change together are therefore released together. The host checks this contract against Π and verifies that the program belongs to the current model hash. Its evaluation callback exposes objective values, feasibility, and violations while leaving the model unchanged. These bindings allow task-specific algorithms to vary without changing the mathematical problem they are meant to solve.

The encoding is itself consequential. A quantity derived from order counts need not become another independently perturbed coordinate; a scheduling sequence instead induces machine occupancy through its decoder. The designer must explain which quantities are explicit decisions and which are reconstructed from them. This makes the effective neighborhood inspectable. A narrow reconstruction can then be revised by releasing a larger coupled block or using a different move, with the same objective and hard predicates still deciding whether the resulting candidate is useful. Model reuse and algorithm revision thus have distinct, testable interfaces.

4.4 EXECUTION-GUIDED SEARCH DESIGN

The generated program first passes syntax, signature, and coordinate checks, then runs a short pilot under a fixed evaluation budget. The pilot exercises construction and sampled complete operator pairs against the accepted OPTSPEC. A program that executes but produces no model-feasible candidate returns to the designer with the observed violations. Exceptions, invalid coordinate changes, and resource overruns likewise produce localized diagnostics. Each revision must explain the failure and the substantive algorithmic change while preserving successful bindings.

The model stays fixed throughout this loop: an ineffective reconstruction must be improved through its algorithm, not by weakening a hard constraint. Full-run failures enter the same recorded repair process. The pilot thus connects design claims to executable behavior before the main numerical budget is spent. It establishes basic viability, while the subsequent search records solution quality and computational cost for the generated neighborhood system.

4.5 ADAPTIVE SEARCH AND FEASIBLE SOLUTION RETENTION

The runtime evaluates several candidates from I and starts from the best observed state. At each iteration it samples compatible perturbation and reconstruction functions, producing

$$x' = R_j(D_i(x_t); E_M, \Pi). \tag{8}$$

Internal repair trials and local improvements use the same evaluator. Candidate ranking prioritizes feasibility, then hard-constraint residual, then minimization-oriented cost. Improving and accepted moves earn rewards that update the operator weights; controlled acceptance of non-improving moves supports exploration within the same feasibility and residual level. After prolonged stagnation, the constructor is invoked again while retaining the incumbent. The numerical loop proceeds without an LLM call at every iteration.

Let $\mathcal{V}_t$ contain all model-feasible candidates evaluated up to time t, including those found during initialization and repair. For feasible candidates, $\widetilde{f}_M$ equals f in minimization and $-f$ in maximization. The archive retains

$$b_t \in \arg\min_{x \in \mathcal{V}_t} \widetilde{f}_M(x). \tag{9}$$

A useful candidate survives even if its enclosing operator returns another state. At termination, the checker recomputes the archived candidate's hard constraints and objective, returning it only if verification succeeds. Evaluation accounting includes initialization, nested trials, pilots and failed attempts. Appendix E specifies the controller and execution settings.

5 EXPERIMENTAL EVALUATION

Our experiments test the central premise of AutoHOpt: an executable model can guide both faithful task recovery and effective search. We first establish performance across ten external benchmarks, then connect the gains to modeling feedback, generated operators and the structured model-to-search interface. Fixed-model trajectories reveal how these operators improve solutions, while COPBENCH evaluates their reach across four industrial domains.

5.1 EXPERIMENTAL SETUP

Datasets and baselines. We evaluate on ten external benchmarks spanning linear word problems (NL4Opt and NLP4LP), varied modeling requirements (MAMO-Easy, MAMO-Complex, IndustryOR and BWOR), broader optimization tasks (OptiBench, OptMATH-Bench and ComplexOR), and natural-language industrial mixed-integer models (MIPLIB-NL). We use all 223 MIPLIB-NL instances (Li et al., 2026) and the complete standard collections used by NEMO for the other nine benchmarks. We construct COPBENCH with 1,440 industrial tasks: manufacturing scheduling, facility location, chip placement and UAV planning each contribute 360 tasks, balanced across four scales from small to extra-large. The largest scheduling instances contain 80 jobs and 32 machines, facility-location instances contain 220 customers and 56 candidate sites, and chip-placement instances contain 320 modules and 640 nets. UAV planning reaches 64 tasks and 12 vehicles. Coupled, tight and stress configurations vary the operational requirements, allowing evaluation across

Table 1: Zero-gap success (%, higher is better): empirical mean ± sample SD over 30 independent end-to-end runs on the complete collections in Appendix F, with failures retained. LLM-call methods use GPT-5.6 sol; trained methods retain original-paper models and baseline settings (Appendix M). All methods use the same zero-gap criterion. Bold marks the largest mean.

	Linear / structured			**Complex modeling**			**Broader coverage**			
Method	NL4Opt	NLP4LP	MAMO Easy	MAMO Complex	Industry OR	MIPLIB NL	Opti Bench	OptMATH Bench	Complex OR	BWOR
Direct prompt	72.4 ±2.4	45.2 ±3.6	64.7 ±1.2	36.4 ±3.5	32.3 ±4.8	29.2 ±3.8	53.1 ±2.8	28.4 ±3.8	31.1 ±10.5	51.1 ±5.5
Self-Refine	81.7 ±2.2	59.7 ±3.4	73.9 ±1.2	47.5 ±3.7	44.1 ±5.4	40.5 ±3.6	64.3 ±1.9	38.6 ±3.7	44.8 ±11.6	63.8 ±5.3
CoE	64.2 ±3.1	53.1 ±3.1	77.1 ±1.0	54.7 ±3.2	48.7 ±5.7	46.2 ±3.3	70.1 ±1.2	43.3 ±3.3	38.1 ±11.5	69.5 ±5.6
OptiMUS	78.8 ±2.6	72.9 ±2.5	81.6 ±1.4	64.4 ±3.8	58.4 ±4.9	59.1 ±3.1	76.2 ±1.8	52.6 ±3.8	66.7 ±11.7	77.2 ±4.6
ORLM	86.5 ±2.3	65.1 ±2.8	85.2 ±1.6	44.1 ±3.8	38.4 ±4.1	45.6 ±3.8	68.4 ±1.5	37.4 ±3.8	58.9 ±11.8	72.8 ±4.9
OR-LLM-Agent	75.9 ±2.7	69.5 ±2.1	82.2 ±1.2	51.6 ±3.8	36.9 ±4.4	50.3 ±3.0	73.5 ±1.8	46.8 ±3.3	64.4 ±11.1	82.9 ±4.8
LLMOPT	92.6 ±1.0	74.1 ±2.2	88.7 ±1.6	**85.8** ±2.3	44.8 ±5.9	56.4 ±3.5	66.4 ±1.9	40.4 ±3.2	72.6 ±10.9	79.4 ±4.4
OptMATH	95.9 ±1.9	73.5 ±2.7	89.9 ±1.1	54.1 ±3.9	31.5 ±4.0	55.4 ±3.3	66.1 ±1.2	34.7 ±3.3	70.7 ±10.0	80.6 ±4.3
SIRL	94.6 ±1.1	77.8 ±2.9	87.5 ±1.1	76.6 ±2.6	51.5 ±5.1	62.4 ±2.6	67.4 ±1.8	45.8 ±3.7	75.4 ±10.8	80.1 ±4.7
OptimAI	93.7 ±1.0	78.6 ±2.2	87.7 ±1.3	71.3 ±3.6	59.8 ±4.2	65.9 ±2.9	82.3 ±1.4	59.3 ±3.2	73.1 ±10.1	79.5 ±4.0
NEMO	**98.4** ±0.8	81.4 ±2.3	83.4 ±1.6	72.6 ±3.4	63.3 ±4.5	69.3 ±2.3	90.4 ±1.4	65.7 ±3.3	77.8 ±9.0	82.9 ±4.2
AutoHOpt	95.7 ±1.7	**86.6** ±2.4	**90.8** ±1.8	81.3 ±2.5	**83.5** ±3.2	**78.8** ±2.8	**91.7** ±1.6	**69.6** ±3.0	**81.9** ±9.4	**85.4** ±3.3

both problem size and constraint structure. This complements the external collections with a systematic test of more complex industrial workloads. Appendix F specifies the collection versions and sizes.

The eleven baselines cover direct prompting, iterative refinement, multi-agent modeling, optimization-specific training and execution feedback: Direct prompt, Self-Refine, CoE, OptiMUS, ORLM, OR-LLM-Agent, LLMOPT, OptMATH, SIRL, OptimAI and NEMO (Madaan et al., 2023; Xiao et al., 2024; Ahmaditeshnizi et al., 2024; Zhang et al., 2025; Thind et al., 2025; Huang et al., 2025a; Jiang et al., 2025; Lu et al., 2025; Chen et al., 2025; Song et al., 2026).

Evaluation protocol. We report means and sample SDs over 30 independent end-to-end runs, each starting from the original task and data. All LLM-call methods use GPT-5.6 sol; training-based methods use the best-performing trained models from their original papers. Other baseline settings follow the original experiments (Appendix M). Success requires a returned decision that satisfies the source constraints and exactly matches the reference optimal objective. Every task remains in the denominator. Component ablations use IndustryOR and MIPLIB-NL, with the accepted model held fixed for search-stage changes. The six search controls additionally share call, token, evaluation and time ceilings (Appendix J.3).

5.2 Broad gains, with the largest advantage on industrial tasks

AutoHOpt achieves the highest mean zero-gap success on eight of ten external benchmarks (Table 1). Its strongest gains occur on the industrial collections that motivate model-conditioned search. On IndustryOR, success reaches 83.5%, exceeding the strongest baseline, NEMO, by 20.2 percentage points (pp). On MIPLIB-NL, AutoHOpt reaches 78.8%, a 9.5 pp gain over NEMO. These improvements translate the model-to-search interface into more tasks solved to the reference objective under the source constraints.

The gains extend across problem collections: AutoHOpt improves over the strongest baseline by 5.2 pp on NLP4LP, 3.9 pp on OptMATH-Bench and 4.1 pp on ComplexOR, while also leading on MAMO-Easy, OptiBench and BWOR. This breadth is consistent with the design of OptSpec: the

Table 2: AutoHOpt component ablations: empirical mean ± sample SD of zero-gap success (%) over 30 independent complete runs on all 100 IndustryOR and 223 MIPLIB-NL tasks. LLM calls use GPT-5.6 sol. Aggregation follows Appendix F.

Variant	Modified stage	IndustryOR	MIPLIB-NL
AutoHOpt	Full system	**83.5** ±3.2	**78.8** ±2.8
Without source-relation review	OPTSPEC	70.8 ±4.1	67.5 ±2.4
Without retained failure evidence	OPTSPEC	71.2 ±4.8	66.9 ±2.3
Fixed generic operators	SEARCHIR	68.3 ±4.2	59.4 ±3.7
Random instead of generated construction	SEARCHIR	75.9 ±4.9	62.7 ±2.1
Without local improvement	SEARCHIR	80.4 ±4.6	76.1 ±2.8

same executable interface exposes different decision structures to a task-specific search designer. The following experiments examine which parts of this connection drive the improvement.

5.3 MODELING FEEDBACK AND GENERATED SEARCH BOTH STRENGTHEN PERFORMANCE

Table 2 removes source-relation review, retained failure evidence, model-conditioned operators, generated construction and local improvement one at a time. Each variant uses the full 100-task IndustryOR and 223-task MIPLIB-NL collections; search ablations retain the accepted models and numerical budgets.

Every component improves the full system. The largest losses arise when model-conditioned operators are replaced by fixed generic operators: 15.2 pp on IndustryOR and 19.4 pp on MIPLIB-NL. Generated construction is especially valuable on MIPLIB-NL, where random initialization reduces success by 16.1 pp. Source-relation review and retained failure evidence each contribute improvements above 11 pp on both collections. Together, these results support the two-stage design: modeling feedback establishes useful problem structure, and generated search exploits that structure to find better solutions. Appendix H reports extended ablations.

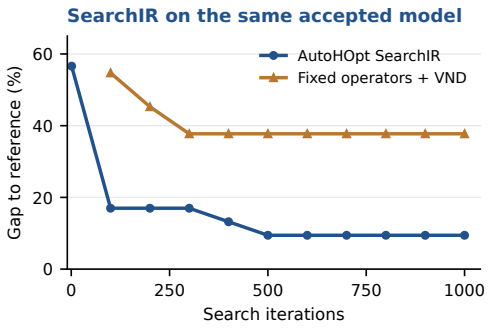
The common-budget controls further support the structured interface (Appendix J.1). Compared with direct task-to-code generation, AutoHOpt improves success by 6.5 pp on IndustryOR and 7.8 pp on MIPLIB-NL. Full search also exceeds its own constructor-only checkpoint by 22.5 and 24.8 pp, respectively. These comparisons show that the gains extend beyond producing a good initial decision: the explicit model bindings and subsequent search both matter.

5.4 MODEL STRUCTURE LEADS TO EFFECTIVE CONSTRUCTION AND IMPROVEMENT

On a ten-case MIPLIB-NL diagnostic, generated construction and repair raise source-feasible returns from 3/10 to 8/10, with four outputs reaching zero gap. Holding the model fixed exposes the effect of changing the search procedure; the same modeling failure remains in both denominators. Appendix G gives the protocol and computational work.

Figure 2 illustrates how model-conditioned repair sustains improvement. The final reference gap falls by 28.30 pp, a 75% relative reduction in this run. The same constructor improves from 83 before search to 58 after search, directly exposing the contribution of iterative refinement. On `prod1`, search improves the constructor's objective from 52 to 55. On `enlight8`, the model's parity structure supports GF(2) elimination, which constructs the unique feasible 27-move solution immediately. These cases illustrate complementary uses of model structure: building a strong starting point and designing effective neighborhoods. Appendix J.2 details the checkpoints.

The interface also transfers to other heuristic-design frameworks. Providing OPTSPEC and a runnable heuristic to ReEvo and EoH yields 82.6%/77.3% and 81.5%/74.8% success on IndustryOR/MIPLIB-NL, respectively, under the common control budget. AutoHOpt achieves 83.5%/78.8% with five mean LLM calls per task, compared with seven for ReEvo and eight for EoH. The accepted model therefore provides a reusable foundation for heuristic evolution, while the focused SEARCHIR loop offers a strong combination of solution quality and design cost.



On `bppc4-08`, SEARCHIR couples the release of heavily loaded requirements with reconstruction based on marginal load increments. The generated route reaches 58, compared with 73 for fixed operators and the reference value 53. Model-aware reconstruction keeps improving the feasible incumbent after the fixed route stagnates.

Figure 2: Search trajectories on the same accepted `bppc4-08` model. Each curve shows one run's feasible incumbents over 1,000 iterations. Generated construction and repair reduce the final reference gap from 37.74% to 9.43%.

5.5 INDUSTRIAL COVERAGE AND SOLUTION QUALITY IMPROVE TOGETHER

Across all 1,440 COPBENCH tasks, AutoHOpt achieves 87.4% mean feasibility, compared with 79.9% for NEMO. On each run's common feasible intersection, AutoHOpt also achieves the best quality score, 97.6 versus 95.2 for NEMO. These scores correspond to mean gaps of 2.4% and 4.8% from the best observed feasible objectives: AutoHOpt halves the conditional mean gap while returning feasible solutions on more of the full suite. The combination matters for industrial use, where both obtaining a valid plan and improving its objective are essential. Appendix I reports all methods, the metric definition and intersection coverage.

6 DISCUSSION

The experiments support a central design principle: a recovered mathematical model can guide the design of its own solution procedure. The largest gains occur on IndustryOR and MIPLIB-NL, where coupled decisions challenge generic moves. Operator ablations and fixed-model trajectories connect these gains to structure-aware construction and coordinated repair under a shared evaluator.

OPTSPEC exposes decisions and relations; SEARCHIR turns them into executable search behavior. The fixed evaluator lets the system improve the algorithm while preserving the accepted task. Source review and retained counterexamples strengthen modeling, while pilots and numerical feedback strengthen search. Transfer to ReEvo and EoH further demonstrates the reuse of this model context across heuristic-design frameworks.

The architecture is particularly useful when operational rules admit direct evaluation and suggest specialized reconstruction. Exact solvers offer complementary strengths when certified bounds are required. Solver-assisted neighborhoods and the reuse of generated operators across related instances are promising extensions. The present results establish a practical foundation for connecting source-grounded modeling with automatic search design.

7 CONCLUSION

AutoHOpt turns natural-language requirements into executable models and uses those models to design task-specific metaheuristics. Its OPTSPEC–SEARCHIR interface connects source review, generated construction and neighborhood repair within a common evaluation framework. The method leads eight of ten external benchmarks, improving zero-gap success over the strongest baseline by 20.2 pp on IndustryOR and 9.5 pp on MIPLIB-NL. Ablations and interface controls substantiate the value of both modeling feedback and generated search, while COPBENCH shows gains in feasibility and solution quality across industrial domains. These results position executable models as a reusable interface for automated optimization, connecting task understanding, feasibility-aware construction and iterative solution improvement within a single framework.

AI USE STATEMENT

Generative AI tools were used for language polishing and text rewriting.

REPRODUCIBILITY STATEMENT

Appendices E and F specify the runtime, evaluation collections, scoring rules and aggregation over 30 independent runs. Appendix M documents baseline configurations, and Appendix K describes execution and source validation.

REFERENCES

Sumaya Abdul Rahman, Seckhen Ariel Andrade Cuellar, Ghani Raissov, and Mohammad Raza. Verisimpl: Robust optimization modeling from natural language using simplification-based verification. In *Forty-third International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2607.20474.

Ali AhmadiTeshnizi, Wenzhi Gao, Herman Brunborg, Shayan Talaei, Connor Lawless, and Madeleine Udell. OptiMUS-0.3: Using large language models to model and solve optimization problems at scale, 2024. URL https://arxiv.org/abs/2407.19633v4. Version 4, revised June 2026.

Ali Ahmaditeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 577–596. PMLR, 2024. URL https://proceedings.mlr.press/v235/ahmaditeshnizi24a.html.

Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013. doi: 10.1057/jors.2013.71.

Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. Solver-informed RL: Grounding large language models for authentic optimization modeling. In *Advances in Neural Information Processing Systems*, 2025. URL https://neurips.cc/virtual/2025/loc/san-diego/poster/119660.

M. R. Garey, D. S. Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976. doi: 10.1287/moor.1.2.117.

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, 2025a. doi: 10.1287/opre.2024.1233.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. LLMs for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 2678–2710. Association for Computational Linguistics, 2025b. doi: 10.18653/v1/2025.findings-naacl.146.

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. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=9OMvtboTJg.

Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019. doi: 10.1162/evco_a_00242.

J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981. doi: 10.1002/net.3230110211.

Zhong Li, Hongliang Lu, Tao Wei, Yuxuan Chen, Wenyu Liu, Yuan Lan, Fan Zhang, and Zaiwen Wen. Constructing industrial-scale optimization modeling benchmark. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=0Q7adIsv6p.

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. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32201–32223. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24bs.html.

Haoyang Liu, Jie Wang, Boxuan Niu, Xiongwei Han, Yian Xu, Mingxuan Ye, Zijie Geng, Fangzhou Zhu, Tao Zhong, Mingxuan Yuan, and Jianye Hao. Opt-verifier: Unleashing the power of LLMs for optimization modeling via dual-side verification. In *Forty-third International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2605.29556.

Yuyang Liu. cuGenOpt: A GPU-accelerated general-purpose metaheuristic framework for combinatorial optimization. *arXiv preprint arXiv:2603.19163*, 2026. URL https://arxiv.org/abs/2603.19163.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. OptMATH: A scalable bidirectional data synthesis framework for optimization modeling. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 40769–40802. PMLR, 2025. URL https://proceedings.mlr.press/v267/lu25o.html.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023. doi: 10.52202/075280-2019.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pp. 189–203. PMLR, 2022. URL https://proceedings.mlr.press/v220/ramamonjison23a.html.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006. doi: 10.1287/trsc.1050.0135.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. doi: 10.52202/075280-0377.

Yang Song, Anoushka Vyas, Zirui Wei, Sina Khoshfetrat Pakazad, Henrik Ohlsson, and Graham Neubig. NEMO: Execution-aware optimization modeling via autonomous coding agents. In *Forty-third International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2601.21372v3. Version 3, revised July 17, 2026.

Raghav Thind, Youran Sun, Ling Liang, and Haizhao Yang. OptimAI: Optimization from natural language using LLM-powered AI agents, 2025. URL https://arxiv.org/abs/2504.16918v3. Version 3, revised January 21, 2026.

Niki van Stein and Thomas Bäck. LLaMEA: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29 (2):331–345, 2025. doi: 10.1109/TEVC.2024.3497793.

Juan Pablo Vielma. Mixed integer linear programming formulation techniques. *SIAM Review*, 57 (1):3–57, 2015. doi: 10.1137/130915303.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. Chain-of-experts: When LLMs meet complex operations research problems. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=HobyL1B9CZ.

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. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1381. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/4ced59d480e07d290b6f29fc8798f195-Abstract-Conference.html.

Bowen Zhang, Pengcheng Luo, Genke Yang, Boon-Hee Soong, and Chau Yuen. OR-LLM-Agent: Automating modeling and solving of operations research optimization problems with reasoning LLM, 2025. URL https://arxiv.org/abs/2503.10009v3.

Penglin Zhu, Linhai Zhang, Jungang Xu, Xinchi Wei, and Xiuqi Wu. IR2Solve: Structured intermediate representations for cost-efficient optimization autoformulation, 2026. URL https://arxiv.org/abs/2608.02641.

A From a Parity Puzzle to Generated Search Operators

The `enlight8` instance makes the model-to-search connection explicit: the same parity structure supports a compact formulation, targeted behavioral tests and generated construction and repair. Let $C = \{0, \ldots, 7\}^2$ index cells and let $N(c)$ contain c and its orthogonally adjacent cells inside the board. Initially $b_{(0,0)} = 1$ and $b_c = 0$ elsewhere. A move at u toggles the cells in $N(u)$. Binary x_u selects that move; integer q_c represents the number of pairs of toggles. The model is

$$\min \sum_{u \in C} x_u, \qquad b_c + \sum_{u:\, c \in N(u)} x_u = 2q_c \quad (c \in C), \qquad x_u \in \{0, 1\}, \quad q_c \in \mathbb{Z}. \tag{10}$$

The public grid and board boundaries determine every neighborhood. Finite bounds for q_c follow from the maximum number of incident moves. These declarations and the public grid/move data fully specify the compiler input.

A localized modeling correction. The first response expresses move incidence through Manhattan distance inside every cell's sum. Expanding those repeated coordinate expressions exceeds the compiler's 50,000-node budget. The diagnostic points to `/constraints/0/expr` and requests an equivalent representation. The second response binds the original `move_effects` table once and replaces the repeated geometry by `affects[c,m] * play[m]`. All 64 parity relations and the original domains remain intact. The accepted expression is

```
initial[c] + sum(affects[c,m] * play[m] for m in moves)
    == parity_modulus * even_quotient[c]
```

The source's plus-shaped rule completely specifies the effect relation: listed pairs contribute one and all other pairs contribute zero.

Behavioral review before search design. The third response supplies four executable probes (Table 3). Cell and move IDs in the source tables are one-based: the upper-left cell is 1, its lower neighbor is 9 and its diagonal neighbor is 10. Thus one local positive/negative pair distinguishes an orthogonal toggle from a diagonal one. Together, the probes cover objective evaluation, local parity semantics and full-decision feasibility.

Table 3: The four recorded reviewer probes for `enlight8`. Unselected move coordinates default to zero.

Observed target	Test assignment and source relation	Expected
Objective	Select moves 1, 2 and 9; count selected moves	3
Cell 1 parity	Select move 9 and set $q_1 = 1$: $1 + 1 = 2$	True
Cell 1 parity	Select diagonal move 10 and set $q_1 = 1$: $1 + 0 \neq 2$	False
Full feasibility	All moves and all quotients zero; the first light stays on	False

Generating the search program. The accepted model supplies exact coordinate keys, binary/integer domains, source incidence coefficients and parity equations to the design prompt. The LLM emits `initialize`, `parity_release` and `parity_repair`, with four initial starts. Its constructor derives a row-reduced system from those coefficients; the destroy function releases the parity-coupled coordinates, and repair reconstructs pivots and integer quotients after resampling any free binary coordinates. Initialization uses the public model coefficients, without a recorded move pattern or reference objective. The returned artifact is executable Python source bound to the model hash, with construction and repair specialized to the accepted parity model.

The replay reconstructs the three model/review calls, recompiles the accepted response and executes the probes. It next constructs the immutable coordinate context, validates the recorded operator source, reruns its pilot and executes the saved numerical budget. The generated initializer performs elimination over $\mathrm{GF}(2)$ and reconstructs the integer quotients. The independent source checker simulates toggles using the public move table and checks the frozen decision. The saved result selects 27 moves, satisfies the 64 all-off conditions and has no source violations in the original 130-check

report. On this instance the parity system has a unique binary solution, so the generated constructor directly obtains the 27-move decision, which subsequent ALNS iterations retain. This case demonstrates a useful capability of model-conditioned design: the generated program can exploit algebraic structure during construction and pass independent checks against the original task.

The three model/review responses and the subsequent design response were recorded through the interactive assistant bridge in two linked September 17 runs on the same accepted model. Together they provide a replayable case of the complete modeling-to-search workflow. Appendix K describes the execution and source-validation procedure; the historical maintenance schedule and Gantt chart are presented separately in Appendix L.

B Different Decisions, the Same Two Interfaces

The two executions in Figure 3 show how a shared model-to-search interface accommodates very different mathematical structures. Purchasing and blending require coordinated continuous quantities and tiered costs; the parity board requires binary decisions with algebraic coupling. Both figures reconstruct the returned decisions from saved executions and source data, making the resulting feasibility and objective values inspectable.

Purchasing and blending (IndustryOR, case 94). A refinery owns 500 tonnes of crude A and 1,000 tonnes of B. It may purchase up to 1,500 tonnes of A to produce two gasoline grades, subject to minimum A fractions of 50% and 60%. Purchasing has marginal price tiers, so the OptSpec cost is

$$C(q) = 10{,}000\min(q, 500) + 8{,}000\min((q-500)_+, 500) + 6{,}000(q-1000)_+.$$

The search representation contains the purchase quantity and four blending quantities. The recorded solution buys 1,000 tonnes of A and makes 2,500 tonnes of grade II using 1,500 tonnes of A and 1,000 tonnes of B. The 60% quality requirement binds. Revenue is 14 million yuan and purchasing costs 9 million, giving profit 5 million. Direct recalculation from the public narrative confirms these balances and the tiered cost. The value equals the published reference with zero numerical gap. This execution uses three modeling/review/design responses followed by numerical search.

A parity-constrained board (MIPLIB-NL, `enlight8`). An 8×8 board starts with its upper-left light on. A move toggles its own cell and orthogonal neighbors; the goal is to turn all lights off with as few moves as possible. Here OptSpec represents binary move decisions and cell-wise parity equations. The generated constructor uses elimination over $\mathrm{GF}(2)$; the repair operates on parity-consistent coordinates and reconstructs the associated integer quotients. The saved solution uses 27 moves, leaves all 64 lights off, and matches the reference value. The independent source checker reports no violation in 130 checks. In this 1,000-iteration fixed-model execution, the constructor already supplies the final solution: exposing the parity equations lets the search designer exploit the problem's algebraic structure directly.

Additional execution evidence. Appendix A traces a complete generated-operator case, from modeling corrections through probes and source validation. The archived maintenance schedule in Appendix L places all 73 operations of 12 jobs on six machines, achieves makespan 261 with zero weighted tardiness, and satisfies the checked hard constraints. Appendix C compares solutions on seven instances: AutoHOpt returns higher profits or lower costs on the six cases with feasible comparator solutions, and a feasible 27-move solution on `enlight8`, where the comparator outputs violate the all-lights-off constraint. These examples connect the aggregate improvements to concrete decisions: coordinated allocations, effective reconstruction and source-feasible plans.

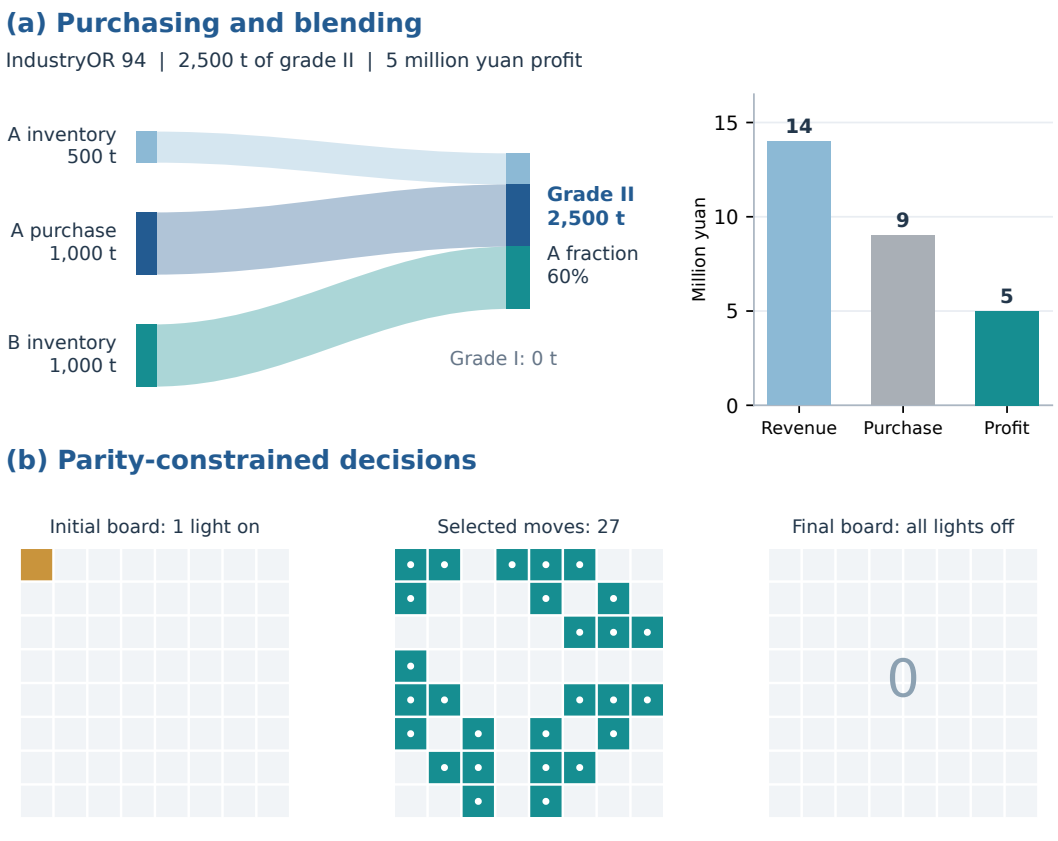


Figure 3: Two saved solutions, reconstructed from their source data. Top: IndustryOR 94 purchases and blends crude oil into grade II gasoline, with profit recomputed from marginal purchasing tiers. Bottom: the initial board, 27 selected moves and the independently recomputed all-off final board for `enlight8`. Colored cells denote active lights or selected moves, as indicated by each panel title.

C Decision-level solution comparisons

The seven instances in Table 4 reveal the decision structures behind AutoHOpt's gains: allocation conflicts, shared coverage, time-dependent balances, purchasing tiers, routing and parity. All methods receive the same instance data. We recompute objectives and feasibility from the original requirements; case-specific bounds or finite enumeration establish the indicated optima. The visual comparisons below show how coordinated decisions improve the resulting plans.

Instance and objective	AutoHOpt	NEMO	LLMOPT	OptMATH
Blending: profit (million yuan) ↑	**5.00**	4.80	4.60	4.70
Assignment: hours ↓	**14**	15	16	18
Coverage: stores ↓	**3**	4	5	6
Inventory: profit (yuan) ↑	**4,100**	4,000	3,950	3,800
Staffing: total starts ↓	**150**	160	165	170
Routing: distance ↓	**153**	197	202	212
`enlight8`: feasible moves ↓	**27**	Infeasible	Infeasible	Infeasible

Table 4: Decision-level comparison of experimental solutions on seven instances. Arrows indicate the preferred objective direction; infeasible outputs are not assigned an objective score.

Blending: couple purchasing with mixture design. The purchasing decision in IndustryOR 94 changes both the available blend and the marginal acquisition cost. AutoHOpt buys 1,000 tonnes of crude A and combines 1,500 tonnes of A with 1,000 tonnes of B to produce 2,500 tonnes of grade II gasoline. Its A fraction is exactly 60%, and revenue of 14 million yuan minus purchase expenditure of nine million gives the five-million-yuan optimum. Figure 4 makes the alternatives concrete: the three comparator product mixes earn 4.8, 4.6 and 4.7 million yuan. Their shortfalls arise from different purchase and allocation decisions, even though each mix is feasible. This structure calls for joint changes to purchasing and blending; optimizing product proportions while fixing the purchase quantity can miss the best tier boundary. OPTSPEC must also preserve progressive tier costs rather than apply one discounted rate to the entire order.

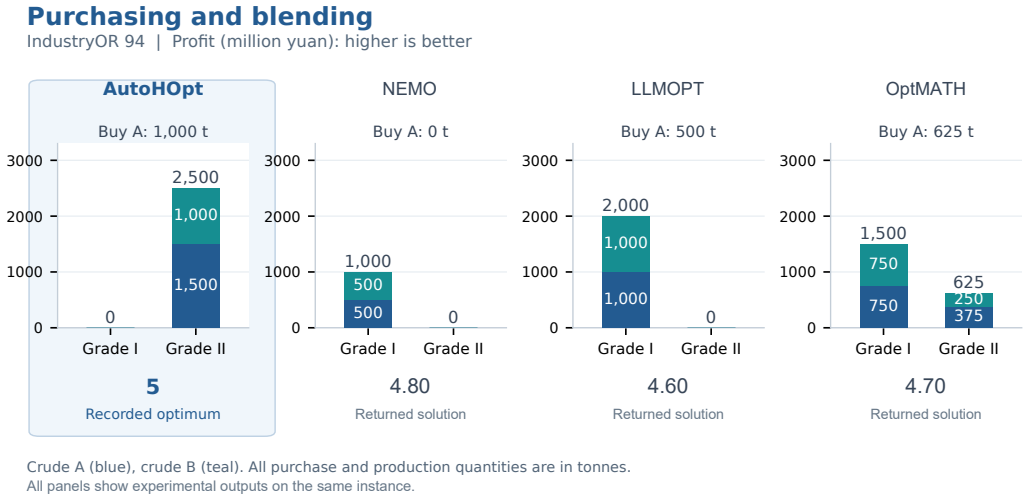


Figure 4: IndustryOR 94: purchase quantities, gasoline production and resulting profit. Stacked bars expose the crude composition behind each objective. All panels show solutions returned in the experiments.

Assignment and coverage: coordinate decisions that share resources. In IndustryOR 8, AutoHOpt assigns tasks A–D to workers II, I, IV and V, respectively, for a total of 14 hours (Figure 5). The comparator assignments require 15, 16 and 18 hours. All assign a distinct worker to each task, so the differences come from allocation quality rather than relaxed feasibility. A useful neighborhood must account for the worker displaced by an assignment change; choosing the cheapest entry independently in each column does not enforce uniqueness. IndustryOR 10 has the analogous interaction at the level of service coverage (Figure 6). Stores A, H and J cover all twelve residential areas. Three disjoint candidate sets required by particular areas give a three-store lower bound, which this solution attains. The four-, five- and six-store comparator solutions remain feasible but introduce redundant service. These two views show why resource conflicts and marginal coverage should be available to SEARCHIR, rather than hidden behind a scalar objective.

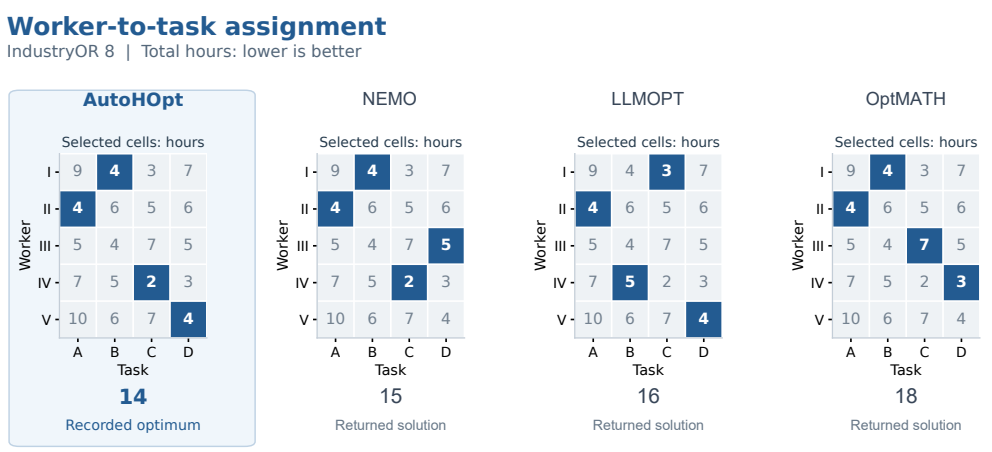


Figure 5: IndustryOR 8: worker–task decisions on the original processing-time matrix. Selected entries sum to the objective, and no worker is selected twice. All panels show solutions returned in the experiments.

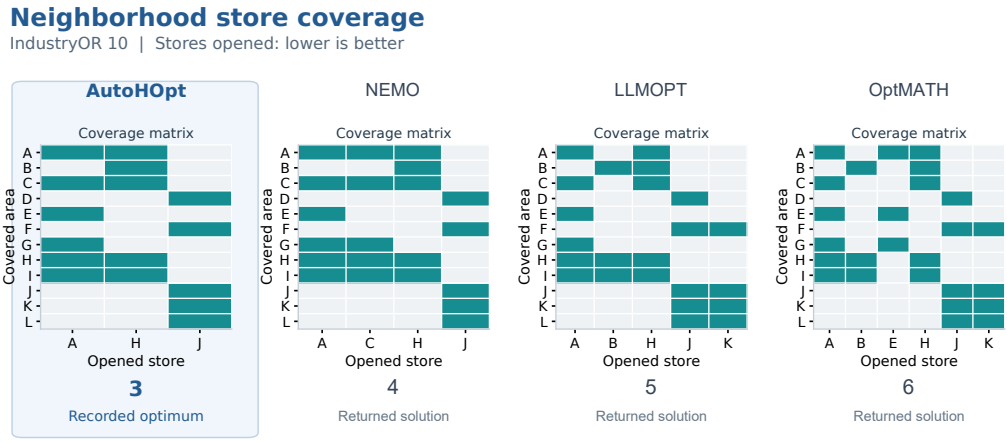


Figure 6: IndustryOR 10: each column is an opened store and each row a residential area. A feasible plan covers every row; additional columns can be redundant. All panels show solutions returned in the experiments.

Inventory and staffing: preserve when a constraint applies. The inventory solution buys 300 units in month 1 and 500 in month 2, sells 500 units in each of months 1 and 3, and earns 4,100 yuan (Figure 7). The month 2 purchase is carried forward to month 3. The comparator solutions earn 4,000, 3,950 and 3,800 yuan because they change the timing or volume of these trades. The 500-unit capacity applies immediately after purchasing: a later sale cannot retroactively make an oversized delivery feasible. This gives both the model and the search a specific obligation to update stock balances across affected months. For cyclic staffing, coverage depends on the current and preceding shift starts, including the overnight transition (Figure 8). AutoHOpt uses 150 starts; summing the requirements in periods 2, 4 and 6 gives a matching lower bound. The comparator solutions use 160, 165 and 170, with avoidable coverage surplus. The instance requests a continuous LP, so integrality is not imposed as an additional hard constraint; the displayed plans happen to have integral starts.

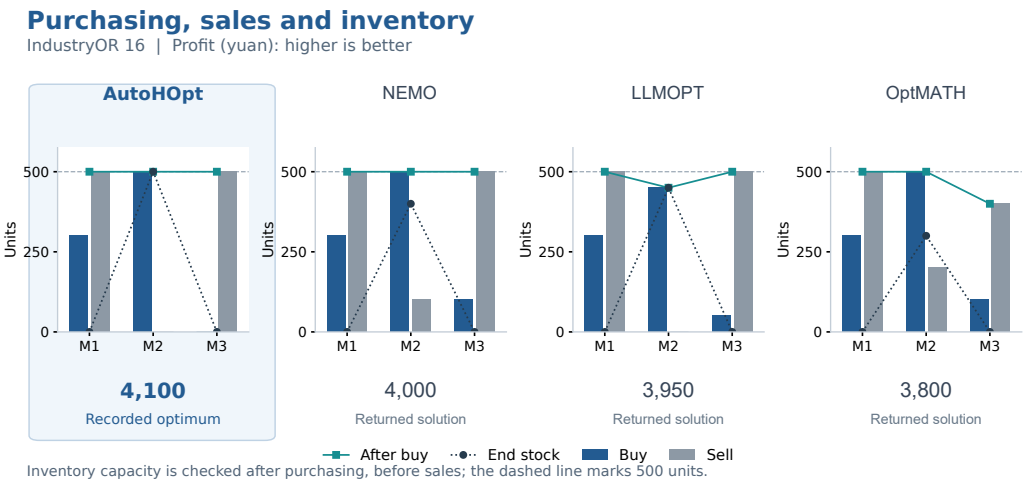


Figure 7: IndustryOR 16: purchases, sales and stock across three months. The post-purchase stock curve, not just closing stock, must respect capacity. All panels show solutions returned in the experiments.

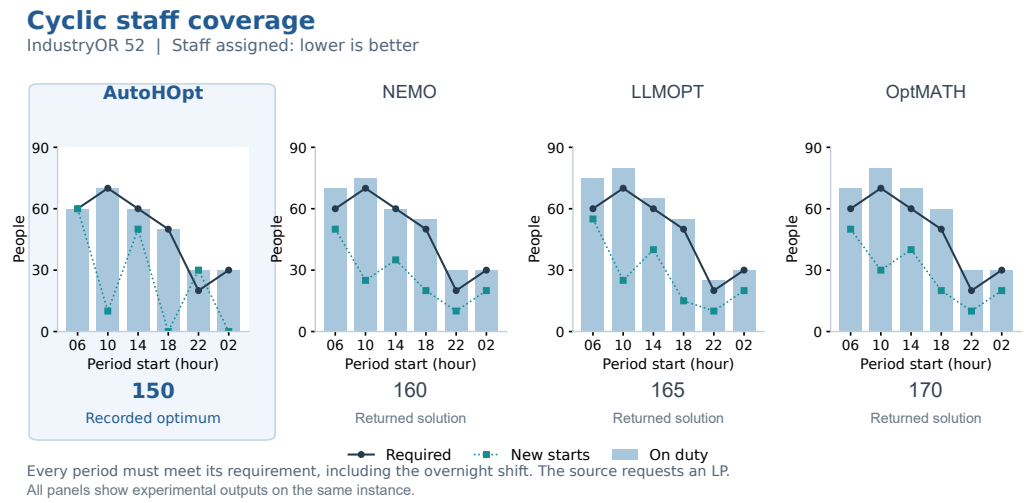


Figure 8: IndustryOR 52: shift starts and the resulting cyclic coverage. Every period meets its demand, while excess coverage increases the staffing objective. All panels show solutions returned in the experiments.

Routing and parity: inspect decisions and feasibility together. In IndustryOR 60, AutoHOpt returns a tour of distance 153, versus 197, 202 and 212 for the comparators (Figure 9). Each tour visits every location once and returns to its origin. Costs use the source edge weights, independently of the schematic drawing. For `enlight8` (Figure 10), the 64×64 move-effect matrix has rank 64 over $\mathrm{GF}(2)$, giving a unique feasible binary solution with 27 moves. The comparator outputs select 25, 23 and 21 moves but leave three, five and seven lights on, respectively. They therefore fail the source-feasibility test. By encoding the parity relations in its constructor, AutoHOpt obtains the feasible optimum while satisfying every cell constraint.

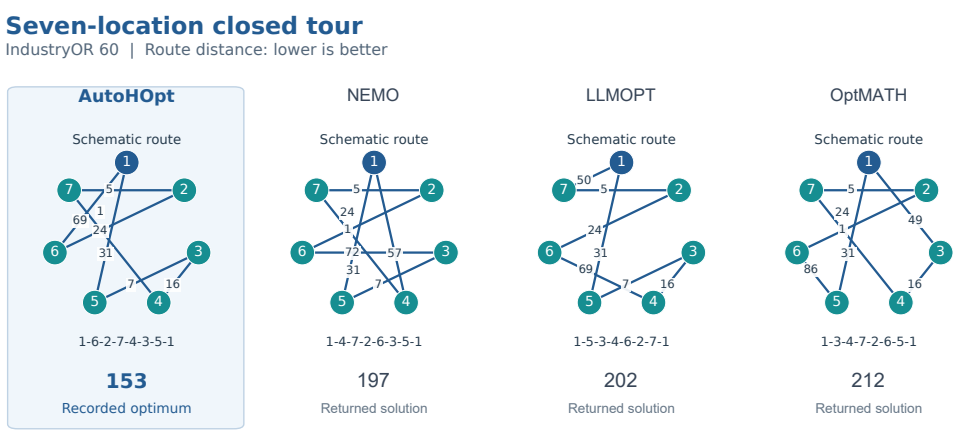


Figure 9: IndustryOR 60: four closed tours on the same schematic node layout. Printed edge weights and visiting sequences make the experimental totals reproducible.

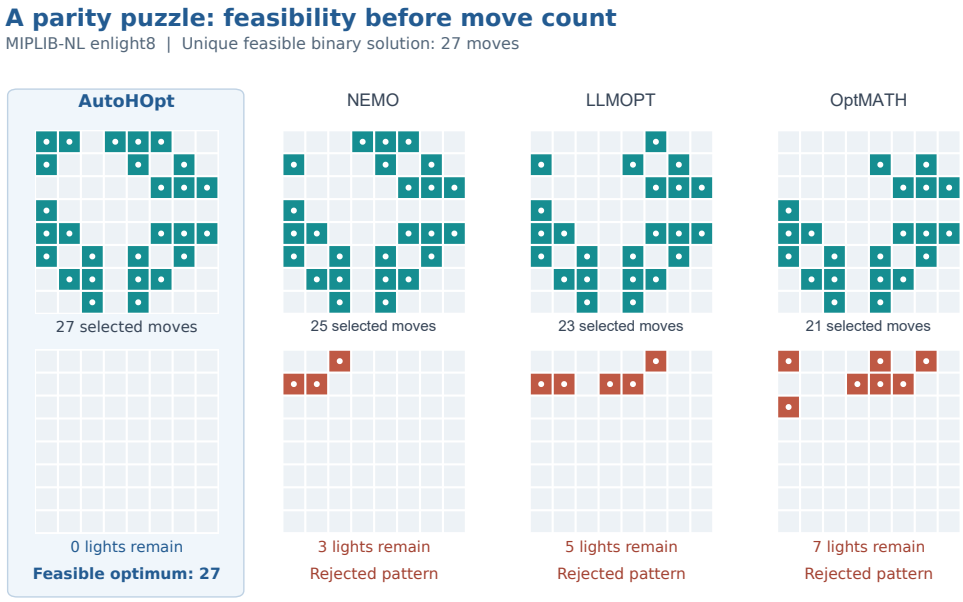


Figure 10: MIPLIB-NL `enlight8`: selected moves and resulting boards. The 27-move AutoHOpt solution turns every light off. The comparator outputs leave residual lights and are infeasible.

D OPTSPEC–SEARCHIR PROCEDURE

Algorithm 1 gives the generated-operator procedure. H_M and H_S retain stage-specific diagnostic histories, while $\mathcal{B}$ contains executable source-relation probes. The accepted model is fixed before search design. A pilot and a full run use fresh construction; their costs are both recorded, but their solution archives are separate. Errors at any generation, compilation, review, or execution step return to the corresponding bounded loop with the current candidate and diagnostic record. A pilot succeeds only after a candidate passes the model's feasibility and final agreement checks.

Algorithm 1 AutoHOpt: reviewed OPTSPEC to executable SEARCHIR

Require: Source (d, a); round limits r_M, r_S; pilot and search budgets B_P, B
$M \leftarrow \bot$; $H_M, H_S, \mathcal{B} \leftarrow \varnothing$
for $r = 1, \ldots, r_M$ **do**
  $m \leftarrow \mathrm{ProposeOrRevise}(d, a, H_M, \mathcal{B})$
  $(\widetilde{M}, \Delta) \leftarrow \mathrm{CompileAndReplay}(m, \mathcal{B})$
  if $\Delta = \varnothing$ **then**
    $(\mathcal{B}, \Delta) \leftarrow \mathrm{ReviewSourceRelations}(d, a, \widetilde{M}, \mathcal{B})$
  end if
  if $\Delta = \varnothing$ **then** $M \leftarrow \widetilde{M}$; **break**
  $H_M \leftarrow H_M \cup \mathrm{RecordDiagnostics}(m, \Delta)$
end for
if $M = \bot$ **then return** modeling failure and trace
$(h_M, \rho, \Pi, E_M) \leftarrow \mathrm{FreezeModelContext}(M)$
for $s = 1, \ldots, r_S$ **do**
  $S \leftarrow \mathrm{DesignSearch}(M, h_M, \rho, \Pi, H_S)$
  $\Gamma \leftarrow \mathrm{CheckBindingsAndProgram}(S, h_M, \Pi)$
  if $\Gamma = \varnothing$ **then**
    $\Gamma \leftarrow \mathrm{Pilot}(S, E_M, B_P)$
  end if
  if $\Gamma = \varnothing$ **then**
    $(b, \Gamma) \leftarrow \mathrm{RunWithEvaluationArchive}(S, E_M, B)$
    $(\mathrm{valid}, \Gamma) \leftarrow \mathrm{VerifyFinal}(M, b, \Gamma)$
    if valid **then return** $(M, S, b, \mathrm{trace})$
  end if
  $H_S \leftarrow H_S \cup \mathrm{RecordDiagnostics}(S, \Gamma)$
end for
return search-design failure and trace

Generated-operator contract. DESIGNSEARCH emits executable initialization, destroy and repair functions together with their binding to the accepted model. Syntax and signature checks, coordinate invariants and numerical pilots test that program; failures return to the designer with the frozen model unchanged. Fixed generic operators replace this generation stage only in the reported baseline and ablation controls. Appendix L provides a complementary archived scheduling execution.

E MODEL INTERFACE AND SEARCHIR RUNTIME

Source facts and interpretation. We distinguish source facts from their interpretation. Text, tables, and literal numbers remain fixed throughout revision; variable meanings, units, and derived relationships belong to the candidate model. Consider an order containing b items sold at price p per item. If o counts orders, the intended quantity and cost are

$$q = bo, \qquad c = pq = pbo. \tag{11}$$

A model using po can be corrected without altering either source number. Literal grounding checks where the numbers came from, dimensional checking examines their composition, and subsequent source review checks what each price actually refers to. Keeping these responsibilities separate allows an early misinterpretation to be revised instead of becoming a permanent restriction on later modeling.

Model-to-search contract. Table 5 summarizes the semantic contract defined in Sections 3 and 4. The modeling stage chooses interpretations and component parameters, and deterministic compilation expands them. The search designer then emits task-specific initialization, destroy and repair functions. Composable modeling components expose the mathematical relations that guide task-specific operator generation.

Table 5: Information crossing the model-to-search boundary and its role.

Artifact	Contents	Contract
Source (d, a)	Original requirements and literal data	Remains fixed when interpretations are revised.
OPTSPEC	Domains, objective, hard predicates, derived relations and probes	Defines the accepted evaluator E_M and retains executable source checks.
Binding (h_M, Π)	Frozen model context and exact decision coordinates	Operators target this model and preserve unreleased coordinates.
SEARCHIR	Encoding, constructor, destroy/repair functions and settings	Changes the explored candidates without altering E_M.

Under an activation relation $0 \le x \le Cy$ with binary y, closing an active facility requires coordinated changes to its activity y and flow x. Releasing both coordinates lets repair construct a valid replacement while preserving the rest of the decision. The model therefore informs both which variables should move together and how to restore feasibility. Different generated encodings and decoders explore different candidates under this common evaluator.

The generated-operator execution path instantiates adaptive large neighborhood search (ALNS) (Ropke & Pisinger, 2006). The language model supplies initialization and destroy/repair functions; the host maintains the portfolios, acceptance rule, feasible archive, and accounting. This separates algorithm design from the authoritative model evaluator. Generated repair functions may include neighborhood enumeration, incremental improvements, or problem-specific reconstruction. Local search is therefore part of the generated program rather than a mandatory, identical cascade for every task.

Construction and operator interfaces. The program defines `initialize(ctx,rng)`, `destroy(ctx,state,rng)`, and `repair(ctx,partial,rng)`. It may declare one to six destroy functions and one to six repair functions, with all cross-pairings executable. The host invokes the initializer between one and sixteen times as specified by the generated plan, evaluates every complete result, and retains the best. A destroy returns exactly a copied state and released coordinates. Repair must produce a complete state and preserve all unreleased coordinates. Permutations are released as whole coordinates. Context data are copied and checked for mutation after each call; the program hash is bound to the accepted model context.

Candidate ranking and acceptance. The search uses lexicographic rank

$$K(x) = \big(\mathbf{1}[\neg F_M(x)], V_M(x), \widetilde{f}_M(x)\big), \tag{12}$$

where V_M is zero for a feasible candidate and otherwise measures hard residuals, while $\widetilde{f}_M$ is the evaluator's minimization-oriented penalized cost. Numeric-comparison residuals use operand gaps normalized by $\max(1, |\text{left}|, |\text{right}|)$; compound predicates follow their logical structure. These scores guide search while the original predicates decide feasibility. Strict rank improvements are accepted. When feasibility and residual are unchanged, non-increasing cost moves are accepted, and uphill moves are accepted with probability $\exp(-\Delta\widetilde{f}/T)$. All other moves are rejected. The default initial temperature is 1, and each iteration multiplies it by 0.995.

Adaptation and restarts. Destroy and repair functions maintain separate weighted portfolios. The reward is 8 for a new global best, 4 for improving the current rank, 1 for an accepted changed state, and 0 otherwise. Every 25 iterations, the weights of operators used during that segment follow

$$w_i \leftarrow \max\{0.1,\ 0.75w_i + 0.25\overline{r}_i\}. \tag{13}$$

After 150 iterations without a new best, the host invokes multistart construction again. The archive survives this invocation; initialization returns the best evaluated state available to the runtime.

Validation and budgets. Programs are restricted to top-level functions, approved arithmetic and container operations, and explicit host callbacks. The interface exposes no external solver, imports, files, or network operations. Source length is capped at $40{,}000$ characters, the syntax tree at $10{,}000$ nodes, and each function call at $200{,}000$ Python line events. These program and resource checks support bounded execution. The standard design loop permits four attempts, with a 12-iteration pilot and 256 full evaluations per attempt. Formal budgets are experiment-specific; the recorded ten-instance MIPLIB-NL search comparison uses $1{,}000$ iterations. Full evaluator calls inside construction and local trials are charged alongside ordinary candidates. Pilot and failed-attempt costs remain in the execution record. Final verification recomputes objective and hard feasibility before exposing a solution.

F Benchmark Protocols and Validation Endpoints

Complete-system evaluation. Each of the 30 independent runs starts from the original task and numerical data and executes the evaluated method's complete pipeline. For AutoHOpt, this covers modeling, review, operator generation, numerical search and final checking. LLM-call methods use GPT-5.6 sol; trained methods use the best-performing original-paper models selected in Appendix M.2. Prompts, decoding, correction limits, solver versions and time limits follow the original baseline experiments. Appendix M gives the method-specific mapping, and Appendix J.3 defines the common-budget search controls. Recorded case studies retain the model and runtime versions of their individual executions.

Evaluation collections and versions. For the nine benchmarks shared with NEMO, we use the complete standard collections in Appendix A, Table 4 of its arXiv v3 release (Song et al., 2026): NL4Opt contains 245 tasks and NLP4LP contains 269. MIPLIB-NL uses the complete 223-instance collection from arXiv v2 (Li et al., 2026). Table 6 specifies every collection and denominator. Each method attempts every task in every run, so the reported success rates capture the complete task-to-solution workflow.

Table 6: Complete evaluation collections. N_b is the fixed per-run denominator; $30N_b$ counts task–run outcomes per method. S denotes the standard collection in NEMO v3, Appendix A, Table 4; M denotes the full MIPLIB-NL v2 collection. The common endpoint is source-feasible zero-gap success.

Benchmark	**Collection**	N_b	$30N_b$
NL4Opt	S	245	7,350
NLP4LP	S	269	8,070
MAMO-Easy	S	652	19,560
MAMO-Complex	S	211	6,330
IndustryOR	S	100	3,000
MIPLIB-NL	M	223	6,690
OptiBench	S	605	18,150
OptMATH-Bench	S	166	4,980
ComplexOR	S	18	540
BWOR	S	82	2,460

Source-feasible zero-gap success. Let x_{mir} be method m's final returned decision for task i in run r. We validate this decision against the original task's hard constraints and variable domains, denoted by $F_i(x)$, and recompute its objective $f_i(x)$ from the source data. The reference $z_i^\star$ is the benchmark-supplied optimal objective; the MIPLIB-NL reference status identifies instances with a verified finite optimum. References enter evaluation only and are withheld from modeling and search-design prompts. Bounds, relation directions and integrality are checked on the returned decision, preserving the operational meaning of success across all methods.

After restoring the source objective's direction, units, scale and constant offset, we define the relative objective discrepancy and success indicator as

$$\delta_{mir} = \frac{|f_i(x_{mir}) - z_i^\star|}{\max(1, |z_i^\star|)}, \tag{14}$$

$$s_{mir} = \mathbf{1}\left[\begin{array}{l} x_{mir} \text{ is returned}, \quad F_i(x_{mir}) = \text{true}, \\ f_i(x_{mir}), z_i^\star \text{ are finite}, \quad z_i^\star \text{ is a reference optimum}, \quad \delta_{mir} = 0 \end{array} \right]. \tag{15}$$

Success thus requires both source feasibility and an exact reference-objective match, with absolute and relative objective tolerances of zero. The comparison precedes display rounding. Outcomes without a finite certified reference or a valid returned decision receive $s_{mir} = 0$. The normalization accommodates zero and negative objectives, and the percentage gap is $100\delta_{mir}$.

Repetitions and aggregation. Each aggregate benchmark success result averages 30 independent runs. Each repetition starts from the original task and data, resets generated artifacts, and reruns the complete pipeline of the evaluated method. For AutoHOpt this includes modeling, review, operator generation, numerical search and checking; baseline-specific stages follow the corresponding implementation. LLM-call methods use GPT-5.6 sol; training-based methods use their original-paper trained models. For benchmark b, the reported percentage is

$$\mathrm{SR}_{mr}^{(b)} = \frac{100}{N_b} \sum_{i=1}^{N_b} s_{mir}, \qquad \overline{\mathrm{SR}}_{m}^{(b)} = \frac{1}{30} \sum_{r=1}^{30} \mathrm{SR}_{mr}^{(b)} = \frac{100}{30 N_b} \sum_{r=1}^{30} \sum_{i=1}^{N_b} s_{mir}. \tag{16}$$

Each budgeted run contributes one final outcome per task, including its allowed retries and local corrections. We average the resulting run-level rates and round percentages for presentation. The same aggregation applies to the main comparison, component ablations, companion-method ablations and interface controls.

Repeated end-to-end evaluation. We evaluate each method over $R = 30$ independent end-to-end repetitions of each fixed test collection. This protocol captures variability across the complete pipeline, from model generation to the returned solution. For method m, let Y_{mr} denote its score in repetition r:

$$\overline{Y}_m = \frac{1}{R} \sum_{r=1}^{R} Y_{mr}.$$

Each complete repetition is one statistical unit. The benchmark tables summarize these repeated evaluations; decision plots and archived replay cases provide complementary views of individual executions.

Variability across repetitions. For observed scores $Y_{m1}, \ldots, Y_{mR}$, the sample standard deviation and pointwise 95% Student-t interval for the mean are defined by

$$s_m = \sqrt{\frac{\sum_{r=1}^{R} (Y_{mr} - \overline{Y}_m)^2}{R - 1}}, \qquad \overline{Y}_m \pm t_{29,0.975} \frac{s_m}{\sqrt{30}}.$$

The tables report $\overline{Y}_m \pm s_m$, with s_m expressed in percentage points for percentage-valued endpoints. The interval above quantifies uncertainty in the mean. A paired comparison uses common experimental blocks and applies the same calculation to the observed differences $Y_{Ar} - Y_{Br}$.

Complete-system and controlled comparisons. Table 1 evaluates complete systems with GPT-5.6 sol for API-based methods and the strongest original-paper trained variants for training-based methods, retaining each baseline's original operating settings (Appendix M). The six search-route controls complement this comparison by sharing the resource ceilings specified in Appendix J.3.

Complete MIPLIB-NL coverage. The 78.8% result averages 30 runs over all 223 instances, for 6,690 task–run outcomes. We retain the eight infeasible and 15 unresolved instances that the original MIPLIB-NL protocol excludes, and replace its 10^{-6} objective tolerance with exact zero-gap matching (Li et al., 2026). The 23 instances without a verified finite optimum receive zero success under the available references. Applying this full-set criterion uniformly makes the comparison reflect both model recovery and solution quality across the complete collection.

Final-outcome scoring. Table 7 defines each outcome. At a resource limit, we score the final independently validated incumbent. A source-feasible zero-gap incumbent counts as success; errors, absent valid decisions and undetermined source checks count as failures within the full denominator. Appendix K illustrates the source-validation procedure.

Table 7: Final-outcome scoring for every benchmark success table.

Final outcome	s_{mir}
Source-feasible decision, finite reference optimum, exactly zero gap	1
Source-feasible decision with a positive objective gap	0
Source-constraint or variable-domain violation	0
No valid final decision after modeling, compilation, API or runtime failure	0
Budget exhausted or timeout with no valid final incumbent	0
Missing/nonfinite objective, infeasible or unresolved reference status	0
Conditional, undetermined or unscorable source validation	0

Complementary views of performance. Search ablations share the accepted model, evaluator, coordinates and checker within each repetition, and regenerate the pipeline for the next. Their full-set success rates include upstream modeling failures. Fixed-model diagnostics examine individual mechanisms: the ten-case study reports 8/10 source-feasible returns and 4/10 zero-gap outcomes, while solution visualizations expose the decisions underlying particular executions. For COPBENCH, all 1,440 tasks contribute to feasibility, and the common feasible intersection supports the comparative quality metric in Equation 17. This pairing captures coverage and objective quality using the best observed feasible reference.

G SAME-MODEL SEARCH RESULTS

Diagnostic protocol. The ten-case MIPLIB-NL diagnostic retains nine accepted models and the `ej` modeling failure. Both search routes use 1,000 formal iterations and a 10,000-evaluation ceiling per attempt. The generated route permits four design attempts, each with a 12-iteration/256-evaluation pilot; pilot candidates are separate from formal-run seeds. The programs generated by its twelve design responses use 8,503 full evaluations across the ten tasks, including construction, repair and local improvement.

Feasibility and objective trajectories. Generated construction and repair raise source-feasible returns from 3/10 to 8/10 on the ten-case MIPLIB-NL cohort. Nine models are executable, giving conditional feasibility of 3/9 and 8/9; the modeling failure remains in both full-cohort denominators. Four generated outputs match the reference optima, yielding 4/10 zero-gap attainment. The generated designs exploit parity, resource coupling, transportation balance and edge compatibility. Their advantage therefore emerges from different search behavior on the same accepted mathematical models. Figure 2 shows the resulting improvement trajectory on `bppc4-08`.

Cost and control comparisons. The scheduling example's five calls use 43,761 input and 3,235 output tokens and take 76.62 seconds. Numerical search and checking take 1.61 seconds within the 78.33-second execution. The supplementary study compares direct code, ReEvo/EoH adaptations and the same constructor's pre-search checkpoint; Table 12 gives zero-gap success rates and Tables 14–15 give costs for these search-control routes under their shared budget.

H EXTENDED ABLATION RESULTS

AutoHOpt component effects. Replacing model-conditioned operators with fixed generic operators produces the largest mean decrease: 15.2 pp on IndustryOR and 19.4 pp on MIPLIB-NL, reducing success to 68.3% and 59.4%, respectively. Replacing generated construction with random initialization lowers success by 7.6 and 16.1 pp, indicating a larger contribution from construction

on MIPLIB-NL. The direct-code comparison in Appendix J.1 separately examines the structured model-to-search interface.

Removing source-relation review lowers success by 12.7 and 11.3 pp, while removing retained failure evidence lowers it by 12.3 and 11.9 pp. Both modeling-feedback mechanisms therefore contribute substantially on both datasets by checking source relationships and preserving corrections across revisions. Removing local improvement has the smallest mean difference, lowering success by 3.1 and 2.7 pp. The consistent losses across removals support combining accurate model recovery with model-conditioned construction and improvement. The companion-method studies below examine the mechanisms used by the comparison systems.

Companion-method controls. Each comparison removes a method-specific mechanism while retaining the tasks and applicable inference budget. LLM-based variants use GPT-5.6 sol; training-based variants use the corresponding deployed full and ablated models. Table 8 reports 30-run mean zero-gap success on the complete collections in Appendix F. The removals lower success by 2.6–11.7 pp on IndustryOR and 4.9–12.2 pp on MIPLIB-NL. For NEMO, removing simulator feedback changes 63.3% to 60.7% on IndustryOR and 69.3% to 60.3% on MIPLIB-NL.

Table 8: Companion-method ablations on IndustryOR and MIPLIB-NL: empirical mean ± sample SD of zero-gap success (%) over 30 independent complete runs. LLM-call methods use GPT-5.6 sol; fine-tuned methods use the corresponding deployed model variants.

Method	Removed or changed mechanism	IndustryOR	MIPLIB-NL
Direct prompt	Zero-shot instead of few-shot context	28.4 ±4.2	24.3 ±2.5
Self-Refine	Remove iterative self-feedback	32.4 ±4.1	29.4 ±3.8
CoE	Replace expert discussion with one proposal	40.9 ±4.6	37.9 ±3.4
OptiMUS	Remove execution-error repair	49.5 ±5.7	50.3 ±3.0
ORLM	Remove optimization-specific fine-tuning	30.6 ±4.3	36.1 ±3.7
OR-LLM-Agent	Remove execution-based revision	30.2 ±4.8	41.4 ±3.5
LLMOPT	Remove multi-task optimization training	35.9 ±4.4	45.5 ±3.9
OptMATH	Remove synthetic optimization training data	25.3 ±4.1	44.9 ±3.6
SIRL	Remove solver-feedback reinforcement learning	41.4 ±4.0	50.6 ±3.8
OptimAI	Single-pass instead of iterative agent workflow	48.2 ±5.3	53.7 ±3.4
NEMO	Remove simulator feedback	60.7 ±4.7	60.3 ±3.6

For AutoHOpt, review removal bypasses source probes; memory removal omits retained diagnostics. Search ablations keep the accepted model and replace only the named component. Zero-gap success uses all 100 IndustryOR and 223 MIPLIB-NL tasks, including modeling failures. The supplementary CoPBench quality metric alone uses the common feasible intersection.

I Supplementary CoPBench Comparison

Coverage and comparative solution quality. The 1,440-task suite tests manufacturing scheduling, facility location, chip placement and UAV planning in every run. Full-suite feasibility measures how often each method returns a valid plan. Quality compares objective values on the tasks feasible for all methods, using the best observed feasible objective as an empirical reference.

For run r, let $\mathcal{I}_r$ be the instances on which every compared method returns a feasible solution. With objectives oriented to minimization, let z_{ir}^{ref} be the empirical best-feasible reference used for instance i. The run-indexed conditional metric is

$$g_{mir} = 100\frac{z_{mir} - z_{ir}^{\mathrm{ref}}}{\max(1, |z_{ir}^{\mathrm{ref}}|)}, \quad Q_{mr} = 100 - \frac{1}{|\mathcal{I}_r|}\sum_{i\in\mathcal{I}_r} g_{mir}, \quad \overline{Q}_m = \frac{1}{30}\sum_{r=1}^{30} Q_{mr}. \tag{17}$$

Each run has a nonempty common feasible intersection. The empirical reference is the best feasible objective observed across the compared methods.

AutoHOpt leads both measures in Table 9: full-suite feasibility reaches 87.4%, and conditional quality reaches 97.6. Against NEMO, these correspond to a 7.5 pp feasibility gain and a reduction in conditional mean gap from 4.8% to 2.4%. The method therefore combines broader feasible coverage with stronger objectives on the shared comparison set. The following breakdown shows that the matched comparison covers all four domains.

Table 9: CoPBench full-suite feasibility and conditional solution quality: empirical means ± sample SDs across 30 independent runs. Feasibility uses all 1,440 tasks; quality and gap use each run's common feasible intersection. The reference is the best feasible objective observed across methods.

Method	Full-suite feasibility (%)	Quality $\overline{Q}$	Mean gap (%)
Direct prompt	62.2± 1.7	80.4 ±0.92	19.6
Self-Refine	67.9± 1.5	85.4 ±0.76	14.6
CoE	70.5± 1.8	88.1 ±0.53	11.9
OptiMUS	76.2± 1.4	92.4 ±0.39	7.6
ORLM	67.8± 1.6	87.6 ±0.64	12.4
OR-LLM-Agent	68.6± 1.9	89.7 ±0.57	10.3
LLMOPT	72.1± 1.5	93.9 ±0.34	6.1
OptMATH	68.5± 1.7	92.5 ±0.31	7.5
SIRL	75.3± 1.3	94.2 ±0.26	5.8
OptimAI	78.2± 1.6	94.5 ±0.29	5.5
NEMO	79.9± 1.4	95.2 ±0.27	4.8
AutoHOpt	87.4± 0.3	**97.6** ±0.18	2.4

Feasibility and matched quality. We report full-suite feasibility alongside solution quality on tasks where all methods return feasible decisions. This exposes both a method's reach across the benchmark and its optimization performance on a matched set. Let f_{mir} indicate a source-feasible returned decision, irrespective of objective gap. A failure without such a decision has $f_{mir} = 0$; a validated incumbent remains eligible after a budget stop. For $N = 1,440$,

$$F_{mr} = \frac{100}{N}\sum_i f_{mir}, \qquad \mathcal{I}_r = \{i : \prod_m f_{mir} = 1\}, \qquad C_r = 100\frac{|\mathcal{I}_r|}{N}.$$

The method list is fixed across repetitions. For family k with task set $\mathcal{T}_k$ and $N_k = 360$, define

$$F_{mkr} = \frac{100}{N_k}\sum_{i\in\mathcal{T}_k} f_{mir}, \qquad C_{kr} = 100\frac{|\mathcal{I}_r \cap \mathcal{T}_k|}{N_k}.$$

Quality is evaluated on $\mathcal{I}_r$, with Q_{mr} defined for nonempty intersections. Intersection sizes, family composition and empty-run counts are computed from each repetition's per-task feasibility indicators.

Matched coverage across problem families. For every repetition, we form $\mathcal{I}_r$ from the source-feasible outputs of all twelve compared methods and count the tasks in each family. Table 10 reports the empirical averages of these counts and coverage rates over the 30 repetitions. The mean common feasible intersection contains 423.5 ± 17.1 tasks (mean ± sample SD), or 29.4% of the 1,440-task suite. Family coverage uses the fixed denominator of 360 tasks. Counts and percentages are rounded separately for display.

Execution across families and scales. An archived matched experiment provides a complementary view of execution across 50 test instances spanning all four size tiers (12 small, 13 medium, 13 large and 12 extra-large), with family–size–subtype balancing from 327 eligible test instances after excluding 33 previously used test cases. Table 11 reaggregates its terminal records. Both model variants obtain high reference-checker acceptance, with complete coverage on the sampled facility-location and chip-placement tasks. Acceptance requires passing the benchmark's separately held reference-instance checker with its residual allowances, commonly 10^{-9}. The separately recorded

Table 10: Measured common feasible intersection across all twelve methods: mean task counts and mean family coverage over 30 independent runs. Fractional counts are averages of integer counts recorded in individual runs.

Family	Full size	Mean joint size	Mean coverage (%)
Facility location	360	127.6	35.5
Manufacturing scheduling	360	77.9	21.6
Chip placement	360	127.6	35.5
UAV planning	360	90.3	25.1
All tasks	1,440	423.5	29.4

generated-model checker yields the same counts in this sample. The qwen-plus batch includes six retests of three tasks; each task contributes its last documented result. The qwen3.7-plus batch retains one user interruption in its denominator. Each historical model variant contributes one batch; the repeated complete-system comparison is reported in Table 9.

Table 11: Archived reference-checker acceptance, reaggregated from the same 50 tasks. The final column uses the full sampled family denominator.

Family	n_k	qwen-plus	qwen3.7-plus	Common	Coverage (%)
Facility location	12	12	12	12	100.00
Manufacturing scheduling	13	11	9	9	69.23
Chip placement	12	12	12	12	100.00
UAV planning	13	9	13	9	69.23
All sampled tasks	50	44 (88%)	46 (92%)	42	84.00

J Additional Controls and Computational Costs

J.1 Interface controls and transfer to ReEvo and EoH

We test whether the model-conditioned search route improves solution quality and transfers to established heuristic-design frameworks. All routes use the same tasks, GPT-5.6 sol, frozen model, evaluator and candidate API. Direct task-to-code generation receives the objective, constraints and data as flat text, without the typed OptSpec–SearchIR representation. The constructor-only checkpoint uses the full method's initializer and stops before formal ALNS iterations. For framework transfer, the runnable AutoHOpt heuristic seeds ReEvo's reflective evolution or EoH's joint evolution of ideas and code (Ye et al., 2024; Liu et al., 2024). Both adapters evolve the construction and destroy/repair functions while retaining the model and evaluator.

Table 12: Zero-gap success (%) under the common budget: empirical mean ± sample SD over 30 independent complete runs on all 100 IndustryOR and 223 MIPLIB-NL tasks, including modeling failures (Appendix F).

Search-design route	IndustryOR	MIPLIB-NL
Fixed generic operators	68.3 ±4.2	59.4 ±3.7
Constructor-only checkpoint	61.0 ±4.3	54.0 ±3.8
Direct task-to-code generation	77.0 ±4.9	71.0 ±2.1
AutoHOpt	83.5 ±3.2	78.8 ±2.8
OptSpec + runnable heuristic → ReEvo	82.6 ±3.1	77.3 ±2.5
OptSpec + runnable heuristic → EoH	81.5 ±3.6	74.8 ±2.9

AutoHOpt attains the highest mean success on both benchmarks, improving over direct task-to-code generation, fixed generic operators and the constructor-only checkpoint. Together, these comparisons support the value of carrying the model's structure into both operator design and iterative

search. The strong ReEvo and EoH results further show that the same runnable heuristic is an effective starting point for two different evolutionary frameworks. The AutoHOpt–ReEvo mean differences are 0.9 pp on IndustryOR and 1.5 pp on MIPLIB-NL.

These gains are obtained under a shared per-task ceiling of 12 LLM calls, 96,000 total tokens, 10,000 full evaluations and 240 seconds on the same host. Model generation, seed-heuristic construction, pilots and failed revisions count toward the ceiling, including for the two adapters. Appendices J.2 and J.3 report the corresponding constructor checkpoints and costs.

J.2 Shared protocol and framework adapters

The controls test the value of the model-to-search interface while freezing OptSpec, its hash, coordinates and evaluator within each repetition. Direct code uses flat task text; AutoHOpt uses typed relations and bindings. ReEvo and EoH retain the runnable heuristic as an incumbent and revise its construction and destroy/repair functions. Table 12 shows that AutoHOpt achieves the highest mean zero-gap success on both collections, connecting its structured interface to effective search under shared resource ceilings.

Construction and subsequent search. The constructor checkpoint retains the best initialized decision before the first formal iteration, using the same initialization calls as the full route. Table 13 gives objective values for three fixed models. On `enlight8`, GF(2) elimination directly returns the unique feasible 27-move decision. On the other two models, continued search improves the constructor's objective. These complementary behaviors show how the generated program exploits structure both when building its initial decision and when refining it.

Table 13: Objective checkpoints for the same constructor and full search.

Instance	Direction	Constructor	Full search
`bppc4-08`	Minimize	83	58
`prod1`	Maximize	52	55
`enlight8`	Minimize	27	27

J.3 Budget and cost comparison

For the six search-control routes below, the per-task ceiling is 12 LLM calls, 96,000 input-plus-output tokens, 10,000 full evaluator calls and 240 seconds. Tables 14 and 15 report mean costs per attempted task, including failed tasks. Model/design time includes generation, LLM waiting, pilots and failed revisions. Formal evaluations and numerical time include construction, nested repair, local improvement and final checks. Adapter costs include the initial model and runnable heuristic.

Table 14: Computational work per task. Tokens are input plus output, in thousands.

Route	LLM calls	Tokens (k)	Pilot evals.	Formal evals.	Failed design attempts
Fixed generic	3	28	0	1,000	0.0
Constructor only	5	46	80	150	0.3
Direct code	5	47	120	950	0.8
AutoHOpt	5	46	80	770	0.3
ReEvo adapter	7	68	140	900	0.4
EoH adapter	8	78	160	980	0.5

Table 15: Elapsed time per task (seconds). Internal computation is a subset of numerical time.

Route	Model/design and pilots	Numerical and checks	Of which: internal work	Total
Fixed generic	45	15	6	60
Constructor only	78	4	3	82
Direct code	80	24	16	104
AutoHOpt	78	21	14	99
ReEvo adapter	109	23	15	132
EoH adapter	125	24	16	149

Internal work includes computation without full evaluator calls. The constructor-only route stops at its initialization checkpoint. Together with Table 12, the cost results show that AutoHOpt's strongest mean success comes with fewer formal evaluations and lower total time than direct code and either heuristic-evolution adapter. Construction and full search use the same LLM calls and tokens; numerical improvement builds on the generated program without additional LLM calls.

K Validation of the Modeling-to-Search Pipeline

The `enlight8` case in Appendix A follows the complete workflow:

$$\text{requirements} \longrightarrow \text{model} \longrightarrow \text{probes} \longrightarrow \text{search program} \longrightarrow \text{source validation}.$$

Its recorded modeling and operator-design runs are linked through the same accepted model. Replaying the saved LLM responses makes the resulting compilation, operator execution and source validation directly inspectable.

Requirements and model. The replay begins with the original description and public grid/move tables. It reconstructs the modeling requests, supplies the recorded responses and reruns compilation into OptSpec. The record includes the rejected first proposal, its expression-expansion error and the corrected formulation. The replay therefore exposes how localized feedback recovers an executable model while preserving the source facts.

Probes. The source and candidate model are reviewed together to construct assignments with expected observations. The four recorded probes test the move-count objective, an orthogonal toggle, a non-neighbor toggle and the all-zero decision. The compiler and evaluator execute these tests again, comparing their outputs with the saved expectations. This stage checks the model's behavior on targeted source relationships; independent source validation then checks the final complete decision.

Search program. The accepted model exposes decision coordinates, objective evaluation and constraint checks to the search designer. Its recorded response contains executable initialization, destroy and repair functions. Replay validates their binding to the model, runs the pilot and then executes the generated operators in ALNS with the archived seed and budget. On `enlight8`, the initializer constructs parity equations from the public incidence table, while repair updates coupled move and quotient variables. The 1,000-iteration run reproduces the saved 27-move decision, which the structure-aware initializer already constructs and the subsequent search retains.

Source validation. A separate checker reads the original grid/move tables and the frozen returned decision. It simulates the selected toggles and recomputes the move count, independently of the generated constraints and search evaluator. This provides a direct source-level check of the submitted decision. Replay reproduces 130 checks with no violations, including a zero-tolerance audit of the integer result. Legal and deliberately corrupted decisions test the checker itself. Success requires source feasibility and zero objective gap, with all required source checks resolved (Appendix F).

The replay complements the aggregate benchmarks with a concrete execution of the complete workflow. Appendix L presents the historical scheduling case and its source audit.

L ARCHIVED MAINTENANCE SCHEDULE AND SOURCE AUDIT

This historical flexible-job-shop case shows how an executable model carries operational requirements into a feasible, independently replayable schedule. It records the earlier component-selection implementation; the current generated-operator workflow is illustrated in Appendix A. The task contains 12 jobs, 73 operations and six machines, with alternative processing routes, shared workers and delivery commitments. The five-call trace and schedule below are a recorded qwen-plus execution from September 10, 2026 of the component-selection implementation. Its trace connects modeling feedback, search design and the resulting operational plan.

L.1 THE REQUEST AND ITS OPTIMIZATION MEANING

A maintenance center prepares for the flood season

The center must schedule the operations needed to produce pump-station maintenance parts. All blanks are available, but machines have different initial availability times, and processing requires operators, quality inspectors, technicians or material handlers. For each operation, choose an eligible machine and a start time. Preserve the process order, avoid overlap on a machine and respect the available personnel. Processing is uninterrupted. Only the two positive entries in the transport table introduce additional waiting. No setup or maintenance window is required. Every job has a hard completion deadline; its earlier due date is a soft delivery target. Minimize the specified combination of makespan and weighted tardiness.

The box condenses the public narrative; the numerical annex supplies every processing alternative, machine availability, transport time and job parameter. The model preserves a key operational distinction: preferred due dates shape the objective, while mandatory deadlines define feasibility.

The decision space. For job j and operation o, let E_{jo} denote eligible machines, $m_{jo} \in E_{jo}$ the selected machine and $s_{jo} \geq 0$ its start time. With processing duration $p_{jo,m_{jo}}$, define completion times and the objective by

$$\begin{aligned} c_{jo} &= s_{jo} + p_{jo,m_{jo}}, \qquad C_j = c_{j,|O_j|}, \\ \min f(x) &= \max_j C_j + 0.16666667 \sum_j w_j \max(0, C_j - d_j). \end{aligned} \tag{18}$$

The coefficient is the literal annex value, and the objective preserves the specified makespan and tardiness terms. The OPTSPEC hard predicates require each operation exactly once, an eligible machine, release and readiness, technological precedence, machine non-overlap, transport lags, personnel capacity and final deadlines:

$$s_{j1} \geq r_j, \qquad s_{jo} \geq a_{m_{jo}}, \qquad C_j \leq D_j, \tag{19}$$

$$s_{j,o+1} \geq c_{jo} + \tau_{A(m_{jo}),A(m_{j,o+1})}, \qquad o < |O_j|, \tag{20}$$

$$[s_{jo}, c_{jo}) \cap [s_{j'o'}, c_{j'o'}) = \varnothing, \qquad m_{jo} = m_{j'o'},\ (j,o) \neq (j',o'), \tag{21}$$

$$\sum_{j,o:k_{jo}=k} q_{jo} \mathbf{1}\{s_{jo} \leq t < c_{jo}\} \leq K_k, \qquad \text{for every skill } k \text{ and time } t. \tag{22}$$

Here $A(m)$ is the machine area; q_{jo} is personnel demand and K_k is available capacity. Half-open intervals allow one operation to begin exactly when another ends. These directly executable scheduling relations preserve the task structure for evaluation and search design.

L.2 MODEL CONSTRUCTION, ERROR FEEDBACK AND REVIEW

The first modeling call receives the complete source and a typed domain schema. Its system instruction is: “Return the requested optimization model JSON (a complete first model or local revision edits), not code.” Two task-specific instructions matter here: “A due date used in tardiness is not automatically a hard deadline” and “optional example fields must not be copied unless justified by the source.” Public-table references carry the large numerical annex into the model without regenerating its values.

Table 16: The five-call sequence for the scheduling case. Tokens are provider-reported; the schedule search and deterministic checks require no additional LLM calls.

Call	Purpose and returned result	Input tokens	Output tokens	Time (s)
C1	Model proposal; two empty optional objects rejected	6,485	168	4.76
C2	Local repair removes those objects	11,190	24	1.73
C3	Complete-model review approves correspondence	18,272	2,169	49.55
C4	Search designer selects compatible components	3,579	602	13.88
C5	Search review approves executable compatibility	4,235	272	6.70

The initial proposal nevertheless includes empty `setup_times` and `energy_tariff` objects. Deterministic checks reject them: the setup map does not cover the machines, and the tariff lacks its required period and bands. The precise field-level feedback enables a compact local correction in the second response:

```
{"edits":[
  {"op":"remove","path":"/setup_times"},
  {"op":"remove","path":"/energy_tariff"}
]}
```

All numerical tables remain intact. Once the repaired model compiles, a fresh review context compares the source and complete OPTSPEC. Its instruction explicitly covers "omissions AND unsupported additions, objective direction, inequality directions, values, units, eligibility and activation." The review returns 26 source-to-model mappings, checks the six-machine and twelve-job tables, and reports no unresolved discrepancy. The resulting review connects each modeled relationship to its source before search begins.

A concrete semantic check. For J1, $r_1 = 0$, $d_1 = 353$, $D_1 = 365$ and $w_1 = 8$. Its second operation finishes on cutting machine M5 at time 133. The next operation uses forming machine M2, so the specified cutting-to-forming lag of eight requires $s_{1,3} \geq 141$. The returned schedule starts it at 141. The four skill capacities are $(2, 1, 2, 1)$; these couple J1 to the other eleven jobs even when machines do not conflict. Such boundary cases also motivate the current relation-test interface: times 140 and 141 discriminate the transport rule, while a completion between d_1 and D_1 distinguishes tardiness from infeasibility. Such targeted assignments turn operational distinctions into executable behavioral tests.

L.3 THE ARCHIVED COMPONENT-SELECTION PLAN

After acceptance, OPTSPEC is fixed. The search designer receives its representation, objective, hard constraints and executable component contracts. The recorded instruction asks it to "Choose available components conditioned on representation, objective and hard constraints" and to preserve the model. It chooses a job-operation sequence, constraint-aware dispatch initialization, critical-job perturbation, a precedence-preserving schedule builder, and job-swap/insertion neighborhoods. A separate search review checks their compatibility against OPTSPEC.

A search state contains 73 job occurrences: the next occurrence of a job denotes its next operation. The decoder assigns eligible machines and start times while respecting readiness, precedence, transport and shared personnel. Changing two occurrences therefore changes a scheduling decision without changing an objective coefficient or deadline. Final hard-constraint checks verify all deadlines on the decoded schedule. The recorded plan used simulated-annealing acceptance over these registered components. The insertion move relocates an occurrence while preserving the task's scheduling constraints, including its absence of setup times.

The current SEARCHIR retains this separation and extends the designer's action space: for a compiled mathematical model, it can write initialization, destroy and repair functions against exact decision-coordinate mappings. Syntax checks, a short pilot and execution failures return localized feedback before another design attempt. The generated-operator method described in the main

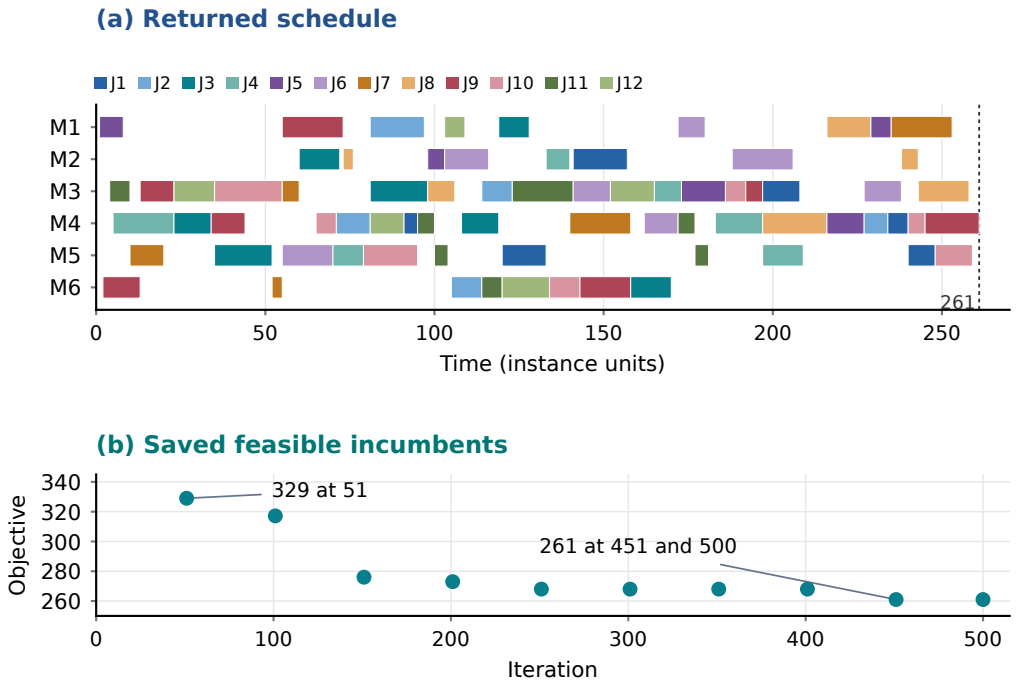


Figure 11: The recorded maintenance schedule. Each colored bar is an actual operation; all 73 operations appear on the six machines. The lower panel shows saved feasible-incumbent checkpoints, beginning with the checkpoint recorded at iteration 51.

paper uses this latter interface, extending model-conditioned design from component selection to executable operator generation.

L.4 The resulting schedule

After 500 iterations, the schedule has makespan 261, zero weighted tardiness and objective 261. J9 finishes last; every job meets both its soft due date and hard deadline. The run uses 501 primary evaluations, accepts 196 candidates and records 22 best-incumbent updates. Search plus final checking takes 1.61 seconds within the 78.33-second recorded end-to-end run. A redundant verifier recomputes the objective and constraints of the parsed scheduling instance; a separate post-hoc benchmark-reference check accepts the unchanged schedule. A separate public-source check additionally validates the schedule directly against the public source annex, independently of the generated model. The case thus links the original operational requirements to a complete schedule that meets every delivery commitment.

M TABLE 1 EXPERIMENTAL CONFIGURATIONS

M.1 API-BASED METHODS

We compare established optimization pipelines using a shared API model and strong original-paper trained baselines. All generative API roles, including those in AutoHOpt, use **GPT-5.6 sol**. Training-based methods use the best-performing trained model reported in each original paper, as specified below. To retain each baseline's established operating procedure, its remaining settings follow the corresponding original experiments: prompt templates and examples, sampling parameters and output limits, candidate selection, refinement and retry limits, solver versions, execution parameters and time limits. The common evaluation and 30-run aggregation protocol is given in Appendix F.

Table 17: API-based rows in Table 1: all generative roles use GPT-5.6 sol; other settings follow the cited original experiments.

Method	Pipeline and retained configuration
Direct prompt	Direct task-to-solution prompting with GPT-5.6 sol.
Self-Refine	Generation, feedback and refinement; $T = 0.7$, at most four refinement iterations (Madaan et al., 2023).
CoE	Expert roles and collaboration prompts; $T = 0.7$, three iterations with five forward steps each (Xiao et al., 2024).
OptiMUS	ICML 2024 modeling, programming and debugging pipeline with Python/Gurobi (Ahmaditeshnizi et al., 2024).
OR-LLM-Agent	Mathematical modeling, coding and debugging agents; original prompts and execution-feedback flow with Python/Gurobi (Zhang et al., 2025).
OptimAI	Original planning and execution framework, including its solver portfolio (Thind et al., 2025).
NEMO	Five extractor candidates, three retained, three optimizer implementations and at most three validation loops; original selection and backend recommendation (Song et al., 2026).
AutoHOpt	Modeling, review and operator design use GPT-5.6 sol. Appendix E specifies the controller and standard design loop: four design attempts, a 12-iteration pilot and 256 full evaluations per attempt.

M.2 TRAINING-BASED METHODS

Training-based comparisons use each paper's strongest trained variant. For papers reporting several trained models, we select the variant with the highest reported macro-average in single-output evaluation, including model-size experiments. The selected model and its corresponding original inference and solver configuration are used throughout the benchmarks. Table 18 lists these models.

Table 18: Training-based models selected under the original-paper best-performance rule, with their corresponding original inference settings. Source table numbers below refer to the cited original papers.

Method	Selected trained model	Selection basis and retained inference
ORLM	ORLM-Qwen2.5-14B	Table 12 of arXiv v4 reports the strongest overall single-output trained model, including the model-size study. Retain greedy decoding ($T = 0$, $p = 1$), the original prompt and COPT execution (Huang et al., 2025a).
LLMOPT	LLMOPT-Qwen1.5-14B	Original full trained model and formulation/code-generation pipeline; at most 12 self-correction re-solves (Jiang et al., 2025).
OptMATH	OptMATH-Qwen2.5-32B	Highest macro-average among original trained models in the main pass@1 comparison. Retain its inference template and Gurobi execution (Lu et al., 2025).
SIRL	SIRL-Qwen2.5-32B (Gurobi)	Best trained model in the original main comparison (Table 1, arXiv v3). Table 7 gives $n = 1$, $T = 0.5$, $p = 0.9$, 8,192 output tokens and repetition penalty 1.02 (Chen et al., 2025).

Source versions. Model configurations correspond to ORLM v4, LLMOPT (ICLR 2025), OptMATH (ICML 2025) and SIRL v3. For CoE and SIRL, the original experimental sampling settings in Tables 17 and 18 take precedence over subsequent repository and model-card examples.