OPTARCHITECT: VERIFIABLE OPTIMIZATION PROBLEM SYNTHESIS VIA RECURSIVE SKILL COMPOSITION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models now solve many optimization-modeling benchmarks at high accuracy, reducing their ability to distinguish stronger systems. Constructing harder problems remains difficult because unconstrained generation weakens correctness, whereas formulation-constrained synthesis limits structural variation. We introduce OptArchitect, which represents modeling mechanisms from public OR resources as executable skills and recursively composes compatible skills through decision-level couplings. Composite skills are admitted only after structural and solver checks, and generated problems are further validated by an independent reconstruction from the natural-language specification. Across four frontier reasoning models, average accuracy decreases from 74.8% on atomic problems to 46.5% on recursively composed problems. The gap remains substantial under size-matched comparisons. OptArchitect-synthesized data also improves downstream optimization modeling when used for fine-tuning.

1 INTRODUCTION

Optimization modeling converts natural-language requirements into mathematical formulations and executable solver programs. Recent large language models have improved rapidly on this task through stronger reasoning, code generation, and tool use. As shown in Figure 1, frontier reasoning models exceed 80% accuracy on most benchmark–model pairs considered in this work, while NL4OPT and MAMO EasyLP are above 95% for all four models. This progress makes several established benchmarks less discriminative for current systems. At the same time, high benchmark accuracy does not imply that practical optimization modeling has become routine. Real applications often combine several decisions through shared resources, states, capacities, and temporal dependencies, and the resulting formulation can differ qualitatively from a collection of isolated constraints.

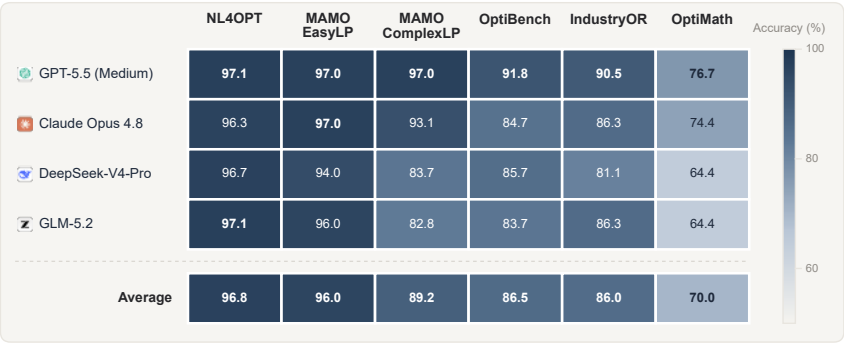


Figure 1: Accuracy (%) of frontier reasoning models on existing optimization-modeling benchmarks. NL4OPT and MAMO EasyLP are above 95% for all four models.

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.

Synthetic data is increasingly used to extend optimization-modeling benchmarks and training corpora. Existing approaches broadly follow two directions. Problem-centric synthesis changes scenarios, objectives, constraints, or parameters of existing tasks, which offers broad semantic variation but may introduce inconsistencies between the statement, formulation, and answer. Formulation-centric synthesis starts from a mathematical program, executable model, or structured modeling process and translates it into natural language, which improves verifiability but tends to preserve the structure of a predefined formulation (Huang et al., 2025; Song et al., 2025; Lu et al., 2025). Public OR resources provide a complementary source of structure. Research papers, competition problems, and established benchmarks contain reusable modeling mechanisms, but these mechanisms are usually embedded inside complete formulations and application-specific narratives.

We focus on the interaction among modeling mechanisms as a source of difficulty beyond raw instance size. Inventory and fulfillment decisions may share the same stock state. Facility-location choices may determine downstream capacity and routing options. Temporal planning may interact with commitment, ramping, or resource constraints. In these cases, one mechanism changes the feasible decisions of another. Increasing the number of variables or constraints within one familiar formulation does not create the same dependency. This distinction motivates a synthesis process that operates on reusable modeling mechanisms and explicitly controls how they become coupled.

OptArchitect represents a modeling mechanism as an executable skill. A skill records reusable operations, structural invariants, transfer conditions, and a solver-certified anchor. Candidate skills are screened for semantic and structural compatibility and then integrated through an explicit bridge between their decision spaces. The bridge is accepted only when removing it changes the feasible decisions or the optimal decisions inherited from the parents. A verified composite is packaged in the same form as an atomic skill and returned to the skill pool, so a composite produced in one round can participate in a later composition. This recursive promotion differs from treating a generated problem as a terminal sample.

Problem synthesis is separated from skill composition. A verified skill is instantiated in a new decision context, producing a natural-language statement, observable data, an executable reference model, and a certified answer. The reference path checks feasibility and solver consistency. A second path sees only the problem statement and data and independently reconstructs the model. A disagreement that admits another defensible interpretation causes the candidate to be rejected or the originating skill to be refined before regeneration. This separation allows structural novelty to be introduced before the final problem is written, while the final natural-language task remains subject to executable and semantic checks.

We construct 116 OptArchitect-Atomic problems and 79 OptArchitect-Composite problems. Across four frontier reasoning models, average accuracy decreases from 74.8% on Atomic to 46.5% on Composite, a drop of 28.3 percentage points. A closed-world study using IndustryOR as the sole source of mechanisms shows that structured recomposition is harder than both atomic reconstruction and random recomposition. Recursive depth is also associated with an increase in the number of variable types, constraint families, and objective terms. After matching solver size, composed problems remain 23.9–38.1 percentage points harder than Atomic problems, reducing the likelihood that the observed gap is explained only by a larger number of variables or constraints. We additionally synthesize training data at multiple recursive depths. Fine-tuning Qwen3-8B on 1K OR-Instruct examples and 1K OptArchitect examples reaches a 71.2% macro average across six downstream benchmarks, compared with 69.2% for SIRL.

We make three contributions.

- We introduce an executable skill representation that separates reusable optimization structure from source-specific context while retaining a solver-certified computational anchor.
- We develop recursive skill composition with compatibility screening and load-bearing decision-level coupling, allowing verified composites to become reusable parents in later composition rounds.
- We construct a verification-guided synthesis pipeline and evaluate the resulting problems as both a benchmark and a source of downstream training data.

2 Related Work

Automated optimization modeling. LLM-based optimization modeling has developed along inference-time and training-time directions. CoE and OptiMUS organize formulation, coding, solver interaction, and revision into structured workflows (Zhong et al., 2024; AhmadiTeshnizi et al., 2024). ORMind introduces structured reasoning and counterfactual analysis, OptiTree uses hierarchical decomposition and tree search, and SolverLLM scales test-time search with solver feedback (Wang et al., 2025; Liu et al., 2026a; Li et al., 2025a). Opt-Verifier and PEARL place verification more directly in the inference loop, using solver signals to identify formulation errors and guide correction (Liu et al., 2026b; Lu et al., 2026). These methods primarily improve how a given optimization problem is interpreted, formulated, or repaired at test time.

Training-based methods address a related question by internalizing optimization knowledge. LM4OPT, ORLM, and LLMOPT use specialized optimization-modeling corpora for supervised fine-tuning or instruction alignment (Ahmed & Choudhury, 2024; Huang et al., 2025; Shu et al., 2025). SIRL, OR-R1, DeepOR, and StepORLM further incorporate solver supervision, reinforcement learning, or process-level feedback (Chen et al., 2026; Ding et al., 2026; Xiao et al., 2026; Zhou et al., 2026). Their performance makes the quality and coverage of optimization-specific training data increasingly important. OptArchitect is concerned with the construction of that data and with the generation of evaluation structures that remain challenging for strong inference-time systems.

Synthetic data generation. General synthetic-data methods have progressed from seed expansion toward more autonomous task construction. Self-Instruct and Evol-Instruct expand instruction sets from existing seeds, while Auto Evol-Instruct learns evolution operators automatically (Wang et al., 2023; Xu et al., 2024a; Zeng et al., 2024). Persona Hub, AgentInstruct, and Magpie broaden task coverage through personas, raw corpora, or model-induced instructions (Ge et al., 2024; Mitra et al., 2024; Xu et al., 2024b). ReverseGen targets failure-inducing instances and Absolute Zero uses self-play with verifiable tasks (Li et al., 2025b; Zhao et al., 2025). Agentic Proposing is especially related in that it composes reusable reasoning skills under tool-based verification (Jiao et al., 2026).

Optimization modeling introduces additional requirements because a synthesized problem must admit a coherent mathematical formulation and a checkable solution. OR-Instruct creates scenario and formulation variants, ReSocratic uses reverse synthesis, and OptMATH starts from mathematical formulations with bidirectional verification (Huang et al., 2025; Song et al., 2025; Lu et al., 2025). MILP-Evolve mutates and crosses MILP problem classes, while Step-Opt increases complexity through iterative modification and problem-level combination (Li et al., 2024; Wu et al., 2025). OptArchitect differs in the unit that is composed and in the criterion used to retain a composition. Its skills encode executable modeling mechanisms rather than complete problem instances. A composite is retained only when the parent mechanisms become decision-dependent, and an accepted composite is promoted as a reusable skill for subsequent composition rather than being treated only as a final generated problem.

3 OptArchitect: Verified Recursive Skill Composition

Figure 2 summarizes OptArchitect. Public OR resources are first converted into executable modeling skills. Compatible skills are then integrated through explicit structural couplings, and composites that pass structural and executable checks are returned to the skill pool. Verified skills are finally instantiated in new decision contexts and checked through reference execution and independent reconstruction. The main text defines the representation and the composition rule. Appendix A gives the complete synthesis procedure, acceptance gates, and implementation details.

3.1 Executable Modeling Skill Construction

A complete optimization problem mixes reusable modeling logic with domain-specific entities, data, assumptions, and narrative. Reusing the full formulation therefore tends to preserve source-specific structure. OptArchitect instead represents a modeling mechanism as

$$s = (\mathcal{O}, \mathcal{I}, \sigma, \mathcal{E}), \tag{1}$$

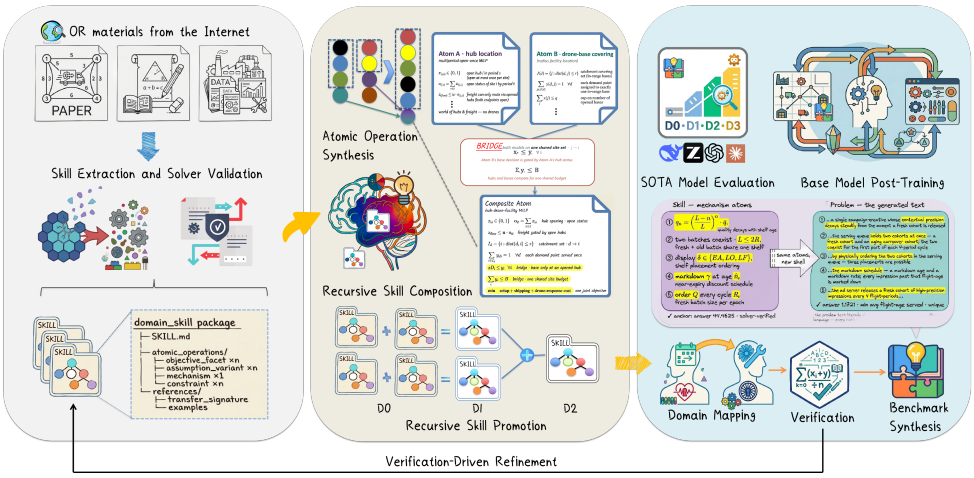


Figure 2: OptArchitect. (a) Public OR resources are distilled into executable skills. (b) Compatible skills are coupled through structural bridges and promoted after verification. (c) Verified skills are instantiated in new decision contexts and checked by dual-view verification.

where $\mathcal{O} = \{o_1, \ldots, o_K\}$ contains reusable modeling operations, $\mathcal{I}$ denotes structural invariants, σ is a transfer signature, and $\mathcal{E}$ is an executable anchor.

Skill representation. The operation set $\mathcal{O}$ decomposes a mechanism into components that can be reused or transferred. These components include a core decision mechanism, constraints, objective variants, and assumption variants. The structural invariant $\mathcal{I}$ specifies the mathematical dependency that should remain unchanged under transfer. For an inventory mechanism, for example, the state relation $I_{t+1} = I_t + a_t - d_t$ may be preserved even when the meanings of inventory, arrivals, and depletion differ across application domains. The transfer signature σ describes the conditions under which the skill can be instantiated, including the required entity types, observable data, decision pattern, objective family, and policy conditions. It defines applicability without fixing a target domain in advance.

The executable anchor is

$$\mathcal{E} = (D, M, y^\star), \tag{2}$$

where D is an instance, M is an executable reference model, and $y^\star$ is its solver-certified answer. The anchor provides a computational witness for the abstract representation. It is used to verify that the skill corresponds to an executable optimization mechanism rather than a textual description whose mathematical consequences cannot be checked.

Skill distillation and certification. Given public OR resources $\mathcal{P} = \{p_1, \ldots, p_N\}$, OptArchitect extracts the decision structure, objective, principal constraints, and mechanism-specific dependencies of each source. The resulting initial pool is

$$\mathcal{S}^{(0)} = \{s_i = \mathrm{Distill}(p_i)\}_{i=1}^{N}. \tag{3}$$

When a source contains stochastic, dynamic, or continuous components that cannot be reproduced directly under the verification environment, we construct a verifiable specialization that preserves the target mechanism under a deterministic executable setting. This step does not replace the mechanism with a generic surrogate. The retained formulation must preserve the dependency that motivates the skill and expose the quantities required by the transfer signature.

Before admission, the anchor must reproduce its certified solution under repeated execution,

$$M_i(D_i) = y_i^\star. \tag{4}$$

The validated formulation is then abstracted into $\mathcal{O}_i$, $\mathcal{I}_i$, and σ_i. Each atomic skill therefore contains both a reusable structural description and at least one executable realization. Appendix A.6 specifies the operation quotas, signature requirements, source-leakage checks, and solver tolerances used for skill admission.

3.2 VERIFIED RECURSIVE SKILL COMPOSITION

A composition is retained only when the parent mechanisms can coexist in one optimization model and interact through the resulting decisions. OptArchitect separates this process into compatibility screening, bridge construction, structural verification, and promotion. The distinction matters because combining two descriptions or two objective terms can increase textual complexity without changing the underlying decision problem.

Compatibility-based skill matching. For two skills s_i and s_j, compatibility is represented by

$$\Gamma(s_i, s_j) = (\gamma_{\text{sem}}, \gamma_{\text{align}}, \gamma_{\text{mech}}, \gamma_{\text{solve}}), \tag{5}$$

where γ_{sem} measures whether the mechanisms admit a coherent joint context, γ_{align} whether their states or decisions can be related, γ_{mech} whether they describe distinct but interacting mechanisms, and γ_{solve} whether their integration remains a single-level executable optimization problem. A useful interface can arise through a shared state, capacity, resource, flow, or temporal variable. Mechanism complementarity is evaluated separately from alignability so that two nearly identical skills are not selected merely because their variables are easy to match.

Pairs that require nested optimal-value feedback are excluded from the standard composition operator. In particular, the solution of one parent cannot be treated as a hidden run-time input that is computed first and then inserted into the other parent. The joint formulation must expose the relevant decisions simultaneously.

Load-bearing structural composition. A compatible pair is composed as

$$s_{ij} = s_i \oplus_{b_{ij}} s_j, \tag{6}$$

where b_{ij} links variables inherited from both parents. The bridge may appear as a constraint

$$b_{ij}(\mathbf{x}_i, \mathbf{x}_j) = 0 \quad \text{or} \quad b_{ij}(\mathbf{x}_i, \mathbf{x}_j) \leq 0, \tag{7}$$

or as an objective relation whose value depends on both decision spaces. Operationally, composition can merge decision spaces, transfer a structural constraint or objective component, or embed one modeling relation inside another. The common requirement is that both parents contribute to the final decision structure.

Consider an inventory skill with transition $I_{t+1} = I_t + a_t - d_t$ and a fulfillment skill with allocation decisions x_{jt}. Replacing the exogenous depletion term with endogenous fulfillment yields

$$I_{t+1} = I_t + a_t - \sum_j x_{jt}. \tag{8}$$

The allocation made in period t now changes the state available to later periods. The two mechanisms therefore cannot be optimized independently once the bridge is introduced. Similar couplings arise when a facility-opening variable activates capacity constraints, when a staffing decision defines service capacity, or when one mechanism creates demand that another mechanism must satisfy.

We verify that the bridge changes the decision problem rather than only the reported objective. Let M_{ij}^b denote the composite model and M_{ij}^{-b} its decoupled counterpart. The bridge is accepted only if removal changes the feasible decision space or the optimal decisions inherited from the parents,

$$\mathcal{F}_{\mathbf{x}}(M_{ij}^b) \neq \mathcal{F}_{\mathbf{x}}(M_{ij}^{-b}) \quad \text{or} \quad \mathcal{X}^\star(M_{ij}^b) \neq \mathcal{X}^\star(M_{ij}^{-b}), \tag{9}$$

where $\mathcal{F}_{\mathbf{x}}$ is the feasible region projected onto the parent decision variables and $\mathcal{X}^\star$ is the optimal solution set. A bridge that changes only an additive accounting term can alter the objective value while leaving every relevant decision unchanged. Such a candidate is rejected. Appendix L.3 gives a concrete case in which an objective-sensitive composition fails this structural test and is regenerated with a tighter coupling.

Recursive skill promotion. An accepted composite is packaged with the same fields as an atomic skill. It has its own operation set, invariants, transfer signature, and executable anchor. Let $\mathcal{S}^{(\leq d)}$ denote all verified skills available through depth d. Candidate composites are

$$\mathcal{C}^{(d+1)} = \{s_i \oplus_{b_{ij}} s_j \mid s_i, s_j \in \mathcal{S}^{(\leq d)},\ \text{Compat}(s_i, s_j) = 1\}. \tag{10}$$

Only verified candidates enter the next level,

$$\mathcal{S}^{(d+1)} = \{s \in \mathcal{C}^{(d+1)} \mid V_{\text{skill}}(s) = 1\}, \qquad \mathcal{S}^{(\leq d+1)} = \mathcal{S}^{(\leq d)} \cup \mathcal{S}^{(d+1)}. \tag{11}$$

We define $\text{depth}(s_i \oplus s_j) = 1 + \max(\text{depth}(s_i), \text{depth}(s_j))$, with atomic skills assigned depth zero. Because a verified composite satisfies the same interface as an atomic skill, it can be matched and composed again at a later depth.

3.3 Verification-Guided Problem Synthesis

A verified skill specifies a reusable optimization structure, not a complete benchmark instance. Problem synthesis maps that structure to a new decision context, produces an executable reference model, and checks whether the natural-language specification implies the same task. We distinguish a transfer, which instantiates one skill in one target context, from an attempt, which is one generate-and-verify cycle within that transfer.

Cross-domain problem realization. Given skill s with transfer signature σ_s, OptArchitect selects a target context whose entities, data, objectives, and policy conditions support the required decision pattern. Let c denote the selected context and ϕ_c the semantic mapping from source entities, data, and decisions to their target counterparts. The mapping is admissible only when the skill invariant is preserved,

$$\mathcal{I}_c = \phi_c(\mathcal{I}_s). \tag{12}$$

A realization produces

$$q = (n_q, D_q, M_q, y_q^\star), \tag{13}$$

where n_q is the natural-language problem, D_q contains observable instance data, M_q is the executable reference model, and $y_q^\star$ is the certified answer. The statement and data are written entirely in the target context, while the formulation retains the invariant relations inherited from the skill. Multiple realizations from the same skill use distinct objective facets so that diversity is not obtained only by changing entity names or domain descriptions.

Problem realization makes functional forms, index conventions, rate parameters, units, and objective aggregation rules explicit when they affect the reference model. The final numerical answer and intermediate solution values remain hidden.

Dual-view verification. Reference execution establishes that M_q is feasible, deterministic, and internally consistent with $y_q^\star$, but executable code can still encode a problem that differs from the natural-language statement. The first path therefore requires $M_q(D_q) = y_q^\star$ within the synthesis tolerance and repeats the solve to detect unstable behavior. The second path receives only (n_q, D_q) and independently constructs $\widetilde{M}_q$, without access to M_q or $y_q^\star$. Let $\widetilde{y}_q = \widetilde{M}_q(D_q)$. We compare the two solutions using

$$\delta(q) = \frac{|y_q^\star - \widetilde{y}_q|}{\max(|y_q^\star|, |\widetilde{y}_q|, \epsilon)}, \tag{14}$$

with $\epsilon > 0$ for numerical stability. If the independent reconstruction supports a materially different answer under a defensible interpretation, the candidate is rejected as ambiguous rather than forcing agreement with the hidden reference implementation. Appendix A.6 reports the independent-probe configuration, adjudication procedure, and tolerances.

Verification-driven refinement. A failed problem is not edited locally to match the reference answer. If f_q denotes diagnostic information from a failed instance, the framework derives a reusable rule $r_q = \text{Generalize}(f_q)$ and updates the source skill as $s' = \text{Refine}(s, r_q)$. A replacement $q' = \text{Generate}(s', c)$ is generated and subjected to the full verification protocol. The update is retained only if the regenerated instance passes without regressing previously satisfied gates. Refinement can therefore affect later transfers from the same skill instead of correcting only the current output. Appendix L.1 shows a two-round example in which successive ambiguity diagnoses update the skill, and Appendix L.2 shows an executable reference model whose code-level error is detected only after independent reconstruction.

After a full problem is accepted, OptArchitect can derive difficulty-controlled variants by masking and restoring modeling components in a nested sequence. Each intermediate problem is independently executable, preserves the narrative context, and is retained only when the restored component changes the resulting decision or objective. Thresholds and an example four-round sequence are reported in Appendix N.

4 EXPERIMENTS

We evaluate four questions. First, we test whether OptArchitect problems are harder than existing optimization-modeling benchmarks for the same models. Second, we isolate the contribution of structured recomposition in a closed-world setting. Third, we examine how model accuracy and formulation structure change with recursive depth. Fourth, we evaluate synthesis reliability and the contribution of individual components. Statistical reliability, cross-backbone synthesis, generation cost, failure analysis, case studies, and downstream training details are reported in the appendix.

4.1 EXPERIMENTAL SETUP

Benchmarks and evaluation sets. We evaluate six existing optimization-modeling benchmarks, including NL4OPT, MAMO EasyLP, MAMO ComplexLP, OptiBench, IndustryOR, and OptiMath. The main comparison reports IndustryOR and OptiMath as representative recent benchmarks, while the complete six-benchmark results appear in Appendix B. OptArchitect-Atomic contains 116 problems realized from individual verified skills. OptArchitect-Composite contains 79 problems realized from recursively composed skills, with 37 problems at D_1, 27 at D_2, and 15 at D_3. Every accepted problem passes the complete verification protocol before entering the evaluation set. No additional filtering is performed after model evaluation.

Models and metric. We evaluate GPT-5.5 (Medium) (OpenAI, 2026), Claude Opus 4.8 (Anthropic, 2026), DeepSeek-V4-Pro (DeepSeek-AI, 2026), and GLM-5.2 (Z.ai, 2026). Agent-based baselines include CoE, Opt-Verifier, OptiTree, and AlphaOpt (Zhong et al., 2024; Liu et al., 2026b;a; Kong et al., 2026); all use GLM-5.2 as the common backbone and receive the same Python and Gurobi access. We report problem-level accuracy. A prediction is correct only when the produced solver code executes and its objective matches the verified reference optimum within the evaluation tolerance. OptArchitect results are averaged over three independent decoding runs. Full inference settings, benchmark filtering, and tolerance definitions are given in Appendix A.

4.2 OPTARCHITECT PRODUCES MORE CHALLENGING OPTIMIZATION PROBLEMS

Table 1 compares the two OptArchitect sets with IndustryOR and OptiMath. Across the four reasoning models, average accuracy is 86.1% on IndustryOR and 70.0% on OptiMath. Atomic problems reduce the average to 74.8%, while Composite problems reduce it further to 46.5%. The Atomic-to-Composite gap is 28.3 percentage points on average and ranges from 17.7 points for Claude Opus 4.8 to 44.7 points for GLM-5.2. The same direction appears for all four agent-based methods. Their average accuracy decreases from 49.5% on Atomic to 25.3% on Composite, even though these systems perform decomposition, search, or verification during inference.

The comparison with all six external benchmarks gives the same context. The four reasoning models average between 84.3% and 91.7% across the existing benchmarks, while the Composite set remains below 60% for three models and reaches only 16.5% for GLM-5.2. Additional stochastic decoding improves Pass@k but does not close the Atomic-to-Composite gap. Appendix E reports three-run standard deviations, Wilson confidence intervals, and Pass@3/Pass@5 results.

Closed-world recomposition. The main benchmark draws mechanisms from several public OR sources, so a harder composed problem could in principle benefit from the introduction of unfamiliar mechanisms. We therefore perform a closed-world experiment using IndustryOR as the sole source pool. Atomic Reconstruction rebuilds problems from individual IndustryOR mechanisms. Random Recomposition samples eligible pairs from the same pool. Structured Recomposition applies compatibility screening and explicit load-bearing bridges. The recomposition settings use the same

Table 1: Accuracy (%) on existing benchmarks and OptArchitect. OptArchitect values are means over three decoding runs. Drop is the decrease from Atomic to Composite.

Method	IndustryOR	OptiMath	Atomic	Composite	Drop
Reasoning LLMs					
GPT-5.5 (Medium)	**90.5**	**76.7**	**84.5**	55.7	28.8
DeepSeek-V4-Pro	81.1	64.4	77.6	55.7	21.9
Claude Opus 4.8	86.3	74.4	75.9	**58.2**	**17.7**
GLM-5.2	86.3	64.4	61.2	16.5	44.7
Average	**86.1**	**70.0**	**74.8**	**46.5**	**28.3**
Agent-based methods (GLM-5.2 backbone)					
CoE	69.5	71.1	54.3	31.6	22.7
Opt-Verifier	66.2	51.7	48.1	21.5	26.6
OptiTree	69.5	63.6	54.3	19.0	35.3
AlphaOpt	68.0	57.8	41.4	29.1	12.3

source pool, depth, number of accepted instances, verification protocol, and approximately matched solver size.

Table 2: Closed-world recomposition of IndustryOR mechanisms. Accuracy in %. Drop is from Atomic Reconstruction to Structured Recomposition.

Method	Original	Atomic Recon.	Random	Structured	Drop
GPT-5.5 (Medium)	90.5	88.4	73.7	60.0	28.4
DeepSeek-V4-Pro	81.1	77.9	61.1	47.4	30.5
Claude Opus 4.8	86.3	85.3	70.5	52.6	32.7
GLM-5.2	86.3	82.1	56.8	35.8	46.3
Average	86.1	83.4	65.5	49.0	34.4

Atomic Reconstruction stays close to the original benchmark, with average accuracy of 83.4% versus 86.1%. Random recomposition lowers the average to 65.5%, while Structured Recomposition lowers it to 49.0%. The source mechanisms are unchanged across these conditions. The remaining difference is therefore associated with how mechanisms are paired and coupled rather than with introducing an external modeling family. We treat this experiment as a diagnostic of structural recomposition, not as a held-out generalization test. The construction details and solver-size statistics are reported in Appendix C.

4.3 Recursive Composition and Structural Complexity

We define D_0 as a problem generated from one atomic skill, D_1 as a composition of two atomic skills, D_2 as a composition containing at least one previously composed skill, and D_3 as deeper recursive composition. Accuracy decreases from 74.8% at D_0 to 56.1% at D_1, 47.3% at D_2, and 21.7% at D_3. The decline is not monotonic for every individual model. DeepSeek-V4-Pro, for example, scores higher at D_2 than at D_1. We therefore interpret depth as an aggregate difficulty axis rather than a deterministic ordering for every model and every adjacent level.

The structural statistics increase with recursive depth. Decision-variable types grow from 3 to 14, constraint families from 4 to 12, and objective terms from 1 to 6. Solver-level model size grows at the same time, so these observations alone cannot separate structural interaction from scale. We address this confound with a size-matched analysis. Atomic problems whose solver sizes fall within 30% of the average D_1 and D_2 sizes are compared with the corresponding composite groups. At the D_1-matched size, Atomic accuracy is 80.0% and Composite accuracy is 56.1%. At the D_2-matched size, the values are 85.4% and 47.3%. The remaining gaps of 23.9 and 38.1 percentage points are consistent with structural coupling contributing difficulty beyond raw solver size. Full per-model depth results, confidence intervals, and the matching procedure are given in Appendices D, D.3, and E.

4.4 Synthesis Reliability and Ablation

A hard benchmark is useful only if low model accuracy is not caused by malformed problems. We evaluate synthesis yield under a controlled protocol in which Direct Generation, Summary Prompting, Atomic synthesis, and Composite synthesis receive the same source corpora, generation backbone, transfer budget, regeneration budget, and acceptance criteria. Direct Generation and Summary Prompting reach overall success rates of 7% and 12%, respectively. OptArchitect reaches 65% for Atomic transfers and 52% for Composite transfers. The accepted evaluation problems are then assessed separately by three OR reviewers. Across correctness, modeling completeness, semantic clarity, practical plausibility, and solution accuracy, every reported positive-rating rate is at least 82%. Composite problems are weaker on completeness and clarity than Atomic problems, at 88% versus 100% and 82% versus 95%, which identifies a remaining cost of integrating several mechanisms rather than hiding it in the aggregate success rate.

Ablation. Replacing structured skills with source summaries produces the largest loss, reducing success from 65% to 15% on Atomic and from 52% to 7% on Composite. Random pair selection reduces Composite success to 22%, and removing the bridge reduces it to 17%. The verification components are also complementary. Without solver verification, independent reconstruction, or skill refinement, Composite success decreases to 22%, 24%, and 30%, respectively.

These ablations separate three roles in the framework. The structured representation preserves the reusable mechanism and executable anchor. Matching and bridge construction determine whether two skills can form a coherent, decision-coupled model. Solver execution and independent reconstruction test different failure modes, while refinement transfers recurring corrections back to the skill. Appendix F reports the controlled generation baselines, Appendix G gives the reviewer protocol and agreement statistics, and Appendix H provides the corresponding analyses.

Downstream utility. We also test whether the synthesized problems provide useful training supervision. Following the ORLM setting, we cluster OR-Instruct by topic keywords to obtain representative application contexts and synthesize additional problems across multiple recursive depths. A Qwen3-8B model is fine-tuned on 2K examples, consisting of 1K OR-Instruct problems and 1K OptArchitect problems. The resulting model reaches 98.0% on NL4OPT, 95.0% on MAMO EasyLP, 68.5% on MAMO ComplexLP, 51.0% on IndustryOR, 70.1% on ReSocratic, and 44.6% on OptiMath. Its macro average is 71.2%, compared with 69.2% for SIRL. The largest gain over SIRL is on MAMO ComplexLP, while OptiMath remains slightly lower at 44.6% versus 45.8%. Training construction and the full comparison are reported in Appendix O.

5 Conclusion

OptArchitect constructs verifiable optimization problems by representing modeling mechanisms as executable skills and recursively composing them through decision-level couplings. The resulting benchmark remains difficult for frontier reasoning and agent-based systems after controlling for solver size, while the verification pipeline maintains executable and semantic checks on accepted problems. The same synthesis process also produces training data that improves average downstream optimization-modeling performance.

AI use statement

Generative AI tools were used solely for translation and language editing to improve the clarity, fluency, and readability of the manuscript. They were not used for methodological development, scientific reasoning, experimental design, implementation, or result interpretation. All AI-assisted revisions were reviewed and approved by the authors, who remain fully responsible for the final manuscript.

REFERENCES

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. Optimus: Scalable optimization modeling with (MI)LP solvers and large language models. In *International Conference on Machine Learning*, volume 235, pp. 577–596, 2024.

Tasnim Ahmed and Salimur Choudhury. LM4OPT: Unveiling the potential of large language models in formulating mathematical optimization problems. *arXiv preprint arXiv:2403.01342*, 2024.

Anthropic. Introducing Claude Opus 4.8. `https://www.anthropic.com/news`, 2026.

Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. Solver-informed RL: Grounding large language models for authentic optimization modeling. *Advances in Neural Information Processing Systems*, 38:106027–106069, 2026.

DeepSeek-AI. DeepSeek-V4 technical report. `https://www.deepseek.com/`, 2026.

Zezhen Ding, Zhen Tan, Jiheng Zhang, and Tianlong Chen. OR-R1: Automating modeling and solving of operations research optimization problem via test-time reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 228–236, 2026.

Tao Ge, Xin Chan, Xiaoyang Wang, Dian Yu, Haitao Mi, and Dong Yu. Scaling synthetic data creation with 1,000,000,000 personas. *arXiv preprint arXiv:2406.20094*, 2024.

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

Zhengbo Jiao, Shaobo Wang, Zifan Zhang, Xuan Ren, Wei Wang, Bing Zhao, Hu Wei, and Linfeng Zhang. Agentic proposing: Enhancing large language model reasoning via compositional skill synthesis. *arXiv preprint arXiv:2602.03279*, 2026.

Minwei Kong, Ao Qu, Xiaotong Guo, Wenbin Ouyang, Chonghe Jiang, Han Zheng, Yining Ma, et al. AlphaOpt: Formulating optimization programs with self-improving LLM experience library. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2389–2400, 2026.

Dong Li, Xujiang Zhao, Linlin Yu, Yanchi Liu, Wei Cheng, Zhengzhang Chen, Zhong Chen, Feng Chen, Chen Zhao, and Haifeng Chen. SolverLLM: Leveraging test-time scaling for optimization problem via LLM-guided search. In *Advances in Neural Information Processing Systems*, volume 38, 2025a.

Qintong Li, Jiahui Gao, Sheng Wang, Renjie Pi, Xueliang Zhao, Chuan Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Forewarned is forearmed: Leveraging LLMs for data synthesis through failure-inducing exploration. In *International Conference on Learning Representations*, 2025b.

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

Haoyang Liu, Jie Wang, Yuyang Cai, Xiongwei Han, Yufei Kuang, and Jianye Hao. OptiTree: Hierarchical thoughts generation with tree search for LLM optimization modeling. *Advances in Neural Information Processing Systems*, 38:120713–120781, 2026a.

Haoyang Liu, Jie Wang, Boxuan Niu, Xiongwei Han, et al. Opt-Verifier: Unleashing the power of LLMs for optimization modeling via dual-side verification. In *International Conference on Machine Learning*, 2026b. arXiv:2605.29556.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. OptiMath: A scalable bidirectional data synthesis framework for optimization modeling. In *International Conference on Machine Learning*, volume 267, pp. 40769–40802, 2025.

Hongliang Lu, Zhong Li, Yuxuan Chen, Yuan Lan, Fan Zhang, and Zaiwen Wen. PEARL: Solver-in-the-loop interactive optimization modeling from natural language. *arXiv preprint arXiv:2607.18256*, 2026.

Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei-ge Chen, Olga Vrousgos, Corby Rosset, Fillipe Silva, Hamed Khanpour, Yash Lara, and Ahmed Awadallah. AgentInstruct: Toward generative teaching with agentic flows. *arXiv preprint arXiv:2407.03502*, 2024.

OpenAI. GPT-5.5 system card. `https://openai.com/index/gpt-5-5-system-card/`, 2026.

Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, Yang Yu, et al. LLMOPT: Learning to define and solve general optimization problems from scratch. In *International Conference on Learning Representations*, volume 2025, pp. 101580–101606, 2025.

Linqi Song, Xiaodan Liang, and Jing Tang. OptiBench meets ReSocratic: Measure and improve LLMs for optimization modeling. In *International Conference on Learning Representations (ICLR)*, 2025.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pp. 13484–13508, 2023.

Zhiyuan Wang, Bokui Chen, Yinya Huang, Qingxing Cao, Ming He, Jianping Fan, and Xiaodan Liang. ORMind: A cognitive-inspired end-to-end reasoning framework for operations research. In *Annual Meeting of the Association for Computational Linguistics*, 2025.

Yang Wu, Yifan Zhang, Yurong Wu, Yuran Wang, Junkai Zhang, and Jian Cheng. Step-Opt: Boosting optimization modeling in LLMs through iterative data synthesis and structured validation. *arXiv preprint arXiv:2506.17637*, 2025.

Ziyang Xiao, Yuan Jessica Wang, Xiongwei Han, Shisi Guan, Jingyan Zhu, Jingrong Xie, Lilin Xu, Han Wu, Wing Yin Yu, Zehua Liu, Xiaojin Fu, Gang Chen, and Dongxiang Zhang. DeepOR: A deep reasoning foundation model for optimization modeling. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(40):34052–34060, 2026.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *International Conference on Learning Representations*, 2024a.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment data synthesis from scratch by prompting aligned LLMs with nothing. *arXiv preprint arXiv:2406.08464*, 2024b.

Z.ai. GLM-5.2: Built for long-horizon tasks. `https://z.ai/blog/glm-5.2`, 2026.

Weihao Zeng, Can Xu, Yingxiu Zhao, Jian-Guang Lou, and Weizhu Chen. Automatic instruction evolving for large language models. *arXiv preprint arXiv:2406.00770*, 2024.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335*, 2025.

Tao Zhong, Jia Zeng, Mingli Song, et al. Chain-of-experts: When LLMs meet complex operations research problems. In *International Conference on Learning Representations (ICLR)*, pp. 48519–48537, 2024.

Chenyu Zhou, Tianyi Xu, Jianghao Lin, and Dongdong Ge. StepORLM: A self-evolving framework with generative process supervision for operations research language models. In *International Conference on Learning Representations*, volume 2026, pp. 6914–6940, 2026.

A Detailed Experimental Setup

A.1 Benchmarks and Data Preparation

We evaluate optimization modeling performance on six existing benchmarks and on the problems synthesized by OptArchitect. For existing benchmarks, we use the original evaluation sets whenever possible. Filtering is performed before evaluation and is independent of model predictions: no instance is removed based on any model output. An instance is removed only if it satisfies at least one of the following criteria: (i) required data are missing from the instance, or entities referenced by the problem statement are not provided; (ii) the instance data are internally inconsistent, for example parameters that contradict the problem statement; or (iii) the reference answer is ambiguous, cannot be verified by an independent solve, or admits multiple defensible optima. Table 3 summarizes the resulting evaluation sets. For MAMO EasyLP, 100 instances are randomly sampled from the 600 original instances with a fixed random seed, and the same filtering criteria are applied after sampling.

Table 3: Statistics of the external optimization modeling benchmarks used in our evaluation.

Benchmark	Original Size	Evaluated Size	Protocol
NL4OPT	245	245	Full set
MAMO EasyLP	600	100	Randomly sampled (fixed seed)
MAMO ComplexLP	211	203	8 invalid removed
OptiBench	> 600	98	100 sampled, 2 invalid removed
IndustryOR	100	95	5 invalid removed
OptiMath	166	90	Missing/ambiguous instances removed

A.2 Evaluation Models

We evaluate four frontier reasoning models: GPT-5.5 (Medium), Claude Opus 4.8, DeepSeek-V4-Pro, and GLM-5.2. Unless otherwise specified, all models receive the same problem statement and observable data and are evaluated under the same answer-extraction and correctness criteria.

Inference configuration. All four models are accessed through their official APIs with the same decoding configuration across OptArchitect evaluations and a maximum output length of 16,384 tokens. For stochastic sampling analyses, we set the temperature to 0.7. Models output the mathematical formulation and solver code in one response and have no interactive access to solvers during generation. We execute the generated code in a sandboxed environment with Gurobi and a 180-second time limit per run. When the generated code fails to execute or times out, the execution error is returned to the model for regeneration. Transient API failures are retried independently and never resample a completed response.

Agent-based baselines. We further compare against CoE, Opt-Verifier, OptiTree, and AlphaOpt. To isolate the effect of the agentic framework from backbone capability, all agent-based methods use GLM-5.2 as the backbone model. We use the official public implementations of the four methods and follow their original pipelines. All agent-based methods receive the same tool access, namely Python execution and Gurobi, and share the same budget of at most 20 agent steps and 10 solver calls per problem.

A.3 Evaluation Metric

We report problem-level accuracy. A prediction is considered correct only when the produced solver code executes successfully and the returned objective value matches the verified reference optimum within a relative tolerance of 5 percent, following the same convention as OptiTree; an absolute tolerance of 0.05 is used when the reference value is zero. When a problem additionally specifies a baseline quantity, the returned baseline must also match. We do not require the full decision vector to coincide with the reference solution, since a problem may admit multiple optimal solutions. For existing benchmarks, we follow the official evaluation scripts when available and apply the above criteria otherwise.

A.4 SYNTHESIS CONFIGURATION

OptArchitect synthesis is implemented as a multi-agent pipeline organized around the three stages described in Section 3. Skill construction agents distill candidate skills from OR sources and certify their executable anchors through repeated solver execution. A composition agent proposes compatible skill pairs and designs structural bridges for candidate composites; proposed composites additionally pass a structural-binding check that rejects couplings altering only the objective accounting without affecting the optimal decisions. Problem realization agents then instantiate verified skills in new decision contexts. Every candidate instance is examined by two independent verification paths: a solver-execution check of the reference model, and an independent reconstruction probe that re-models and re-solves the problem from its natural-language statement alone. The probe is instantiated with Gemini 3.5 Flash and is the only component that uses this model; this model is used neither as the generation backbone nor as an evaluation model, so the generator and the probe do not share inductive biases and the verification gate does not favor any model evaluated on the resulting benchmark. Discrepancies are adjudicated by a neutral judging step. Failed instances trigger a refinement agent that generalizes the failure into a reusable rule and updates the originating skill before regeneration.

Unless otherwise specified, all synthesis agents in the main experiments use GLM-5.2 as the generation backbone. The pipeline can also be instantiated with alternative generation backbones; a cross-backbone analysis is reported in Appendix I. Per-stage token consumption and wall-clock time of the pipeline are reported in Appendix J.

A.5 SYNTHESIS PROCEDURE

Algorithm 1 summarizes the complete synthesis procedure with the notation of Section 3. Deterministic steps, namely pool enumeration, pair pre-filtering, structural delivery checks, and solver verification, are executed by fixed scripts, so they are reproducible; semantic steps, namely compatibility scoring, bridge design, composition, realization, and refinement, are executed by agents. Two implementation choices matter for reproducibility. First, candidate pairs are selected by an absolute score threshold rather than a top-K ranking, so adding skills to the pool never changes whether an existing pair is composed. Second, when a quality gate fails, the responsible agent is re-driven to rewrite its own output; outputs are never repaired by hand, and bridge bindings that change only the objective value without affecting the optimal decisions are rejected and re-composed rather than accepted.

In our runs, each atomic skill is realized in three transfer contexts with pairwise distinct objective facets, and each composite skill is realized in one transfer context.

A.6 VERIFICATION AND ACCEPTANCE PROTOCOL

Acceptance is organized as a four-level hierarchy: skill-level gates admit skills into the pool, composition-level gates admit composites and promote them for further composition, problem-level gates admit synthesized problems into the evaluation sets, and sequence-level gates admit difficulty-controlled problem sequences. Substantive acceptance failures enter the refinement loop of Section 3.3 rather than being repaired at the instance level. Deterministic gates are executed by fixed scripts, and semantic gates are executed by agents; unless explicitly noted, the criteria below are hard acceptance conditions. The empirical distribution of rejected candidates and the corresponding recovery paths are reported in Appendix K.

Skill-level acceptance. A distilled skill is accepted only if it satisfies all of the following. The skill package is complete, with the operation set, structural invariants, transfer signature, and executable anchor of Equation equation 1 all specified. The operation set contains at least six atomic operations and satisfies the required operation-kind quotas, with at least three distinct objective facets, at least two assumption variants, and at least one core mechanism, so that one skill supports multiple genuinely different realizations. The transfer signature declares all required entities, data evidence, and unsuitable domains without naming any target domain. The executable anchor solves deterministically under repeated execution and reproduces its certified answer within a relative tolerance of 10^{-4}. No source-domain identifiers, paper references, or solver names leak into any skill carrier.

Algorithm 1 Recursive Verified Skill Composition and Problem Synthesis

Require: OR corpus $\mathcal{P} = \{p_1, \ldots, p_N\}$; compatibility gate Compat; skill verification gate V_{skill}; depth budget $d_{\max}$; realizations per skill n_s; refinement budget R; answer tolerance δ
Ensure: Verified skill library $\mathcal{S}$ with depth labels; verified problem set $\mathcal{Q}$

Stage 1: executable skill construction
for each source item $p_i \in \mathcal{P}$ **do**
  $s_i \leftarrow \mathrm{Distill}(p_i) = (\mathcal{O}, \mathcal{I}, \sigma, \mathcal{E})$
  if the executable anchor $\mathcal{E}$ reproduces its certified answer under repeated execution **then**
    $\mathcal{S}^{(0)} \leftarrow \mathcal{S}^{(0)} \cup \{s_i\}; \quad \mathrm{depth}(s_i) \leftarrow 0$
  end if
end for
Stage 2: recursive composition
for $d = 1, \ldots, d_{\max}$ **do**
  for each candidate pair (s_i, s_j) from $\mathcal{S}^{(\leq d)}$ with $\mathrm{Compat}(s_i, s_j) = 1$ **do**
    $(\mathrm{mode}, b_{ij}) \leftarrow \mathrm{DesignBridge}(s_i, s_j)$ ▷ borrow constraint, borrow objective, embed formula, or merge variables
    $s_{ij} \leftarrow s_i \oplus_{b_{ij}} s_j$ ▷ primary operations kept, borrowed operations remapped, seed instance written
    if $V_{\mathrm{skill}}(s_{ij}) = 1$ **then** ▷ structural checks, solver check on the seed, bridge-decoupling test
      $\mathcal{S}^{(d+1)} \leftarrow \mathcal{S}^{(d+1)} \cup \{s_{ij}\}; \quad \mathrm{depth}(s_{ij}) \leftarrow 1 + \max(\mathrm{depth}(s_i), \mathrm{depth}(s_j))$
    else if the bridge changes only the objective value without affecting the optimal decisions **then**
      re-drive the composition with a tighter structural coupling
    end if
  end for
  $\mathcal{S}^{(\leq d+1)} \leftarrow \mathcal{S}^{(\leq d)} \cup \mathcal{S}^{(d+1)}$
end for
Stage 3: verification-guided problem synthesis
for each skill $s \in \mathcal{S}^{(\leq d_{\max})}$ **do**
  for $r = 1, \ldots, n_s$ with pairwise distinct objective facets **do**
    $c \leftarrow \mathrm{SelectContext}(\sigma_s)$ ▷ target domain satisfying the transfer signature
    $t \leftarrow 0$
    repeat
      $q \leftarrow \mathrm{Realize}(s, c)$ ▷ problem statement, instance data, reference model M_q, certified answer $y_q^\star$
      $\widetilde{y}_q \leftarrow \mathrm{Reconstruct}(n_q, D_q)$ ▷ independent probe; does not observe M_q or $y_q^\star$
      if q fails the solver check, the answer-uniqueness check, or the delivery gate **then**
        $s \leftarrow \mathrm{Refine}(s, \mathrm{Generalize}(f_q)); \quad t \leftarrow t + 1$ ▷ skill-level refinement, at most R rounds
      end if
    until q passes dual-view verification and all delivery gates, or $t > R$
    if q is accepted **then** $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{q\}$
    end if
  end for
end for
return $\mathcal{S}$, $\mathcal{Q}$

Composition-level acceptance. A candidate composite must pass five gates. First, the selection gate: the pair passes the four-dimension compatibility assessment of Equation equation 5 and an absolute score threshold, so that adding skills to the pool never changes whether an existing pair is selected. Second, the bridge gate: the composite declares at least one explicit cross-skill dependency, with every borrowed operation annotated by its variable remapping, and both parents contribute at least one operation. Third, the structural-binding gate of Equation equation 9: removing the bridge must change the feasible decision space or the optimal decisions of the parent variables; a bridge that changes only the objective value is rejected and the composition is re-driven. Fourth, the executable gate: the composite seed model solves deterministically, and the merged transfer signature

is the union of the parent signatures with intersected unsuitable domains. Fifth, the promotion gate: an accepted composite is packaged in the same form as an atomic skill, with its own operations, invariants, signature, and anchor, and re-enters the pool as a valid parent.

Problem-level acceptance. Every candidate problem passes the problem-level gates of Figure 3, which operationalize the acceptance definition of Section F.2. The gates are grouped into four dimensions. Fidelity gates check that the realization preserves the skill: the semantic mapping covers the required entities, the structural invariants are covered in the formulation and the model, and the data support the stated mechanisms. Soundness gates check that the instance is mathematically well-behaved: the reference model is deterministic under repeated runs, the answer moves under data perturbation, and no materially different defensible solution remains after independent reconstruction and adjudication. Quality gates check the specification: the statement is a complete, self-contained problem with the data presented in-line, and the task requires substantive modeling rather than a lookup. Leakage gates check that neither the statement nor the data schema exposes the answer, the solution path, or any derived quantity. A gate failure triggers refinement and regeneration rather than a local repair of the failed instance. As the sole recorded exception, problems whose model class is provably insensitive to data perturbation record this property instead of failing the perturbation gate. At the transfer-set level, realizations from the same skill must use distinct objective facets, which prevents superficial reskinning across the realizations of one skill.

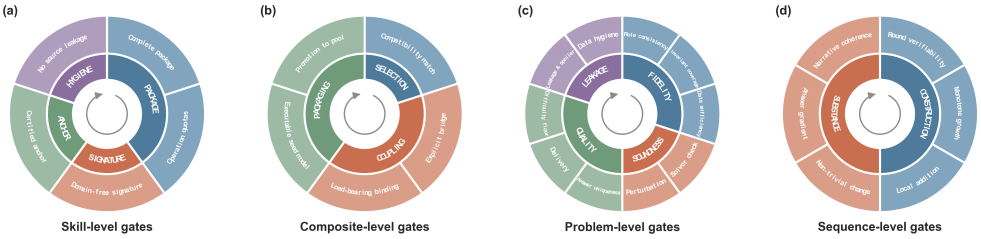


Figure 3: The acceptance architecture of OptArchitect. Gates are organized at four levels: skill-level admission into the skill pool, composition-level admission and recursive promotion, problem-level admission into the evaluation sets, and sequence-level admission of difficulty-controlled sequences. For each level, the inner ring lists the checked dimensions and the outer ring lists the corresponding acceptance gates; a failure at any gate enters the refinement and re-verification loop of Section 3.3.

Difficulty-sequence acceptance. A difficulty-controlled sequence is accepted only if every round is independently verifiable, the set of active modeling components grows monotonically with one to three new components per round, each round changes the resulting decision or objective, the narrative remains coherent across rounds, and the answer gradient satisfies the construction thresholds: the relative difference between the seed and the full problem is at least 30 percent, adjacent rounds differ by at least 5 percent, and the core mechanisms are introduced no earlier than the middle rounds, so the seed does not leak the full structure.

Refinement and audit. A failed verification enters a closed loop: the failure is diagnosed, generalized into a reusable rule that records the failure type, the general principle, and why it generalizes, the originating skill is updated, and the instance is regenerated and fully re-verified; the refinement is kept only if the regenerated instance passes without regressing previously satisfied gates, and is rolled back otherwise. Two audit rules apply to all accepted artifacts. First, an accepted problem package is frozen; the model, data, and certified answer are never edited after acceptance. Second, outputs are never modified to pass a gate; a failed output is rewritten by the responsible agent and re-verified.

Tolerances. The synthesis-time solver check and the evaluation metric use different tolerances for different purposes: the solver check requires the reference model to reproduce its own certified answer within 10^{-4} under repeated execution, which certifies self-consistency of the synthesized

instance, whereas model evaluation compares predictions against the certified optimum within the 5 percent tolerance of Section A.3, which scores the accuracy of an external solver.

B Performance on Existing Optimization Benchmarks

Table 4: Accuracy (%) of frontier reasoning models on existing optimization modeling benchmarks. MAMO subsets are abbreviated as EasyLP and ComplexLP.

Method	NL4OPT	EasyLP	ComplexLP	OptiBench	IndustryOR	OptiMath
GPT-5.5 (Medium)	97.1	97.0	97.0	91.8	90.5	76.7
Claude Opus 4.8	96.3	97.0	93.1	84.7	86.3	74.4
DeepSeek-V4-Pro	96.7	94.0	83.7	85.7	81.1	64.4
GLM-5.2	97.1	96.0	82.8	83.7	86.3	64.4

Frontier models exceed 94% on NL4OPT and MAMO EasyLP and remain comparatively strong on IndustryOR and OptiMath. The saturation pattern motivates evaluation problems with new modeling interactions rather than additional instances of familiar structures.

C Closed-World Recomposition Analysis

We use IndustryOR as the sole source of modeling mechanisms to test whether recombination within a fixed source pool can increase difficulty. Atomic Reconstruction rebuilds single-mechanism problems. Random Recomposition samples eligible pairs uniformly, while Structured Recomposition uses compatibility-based pairing and load-bearing bridges. The recomposition settings use the same source pool, depth, number of accepted instances, and verification protocol and are approximately matched in solver size.

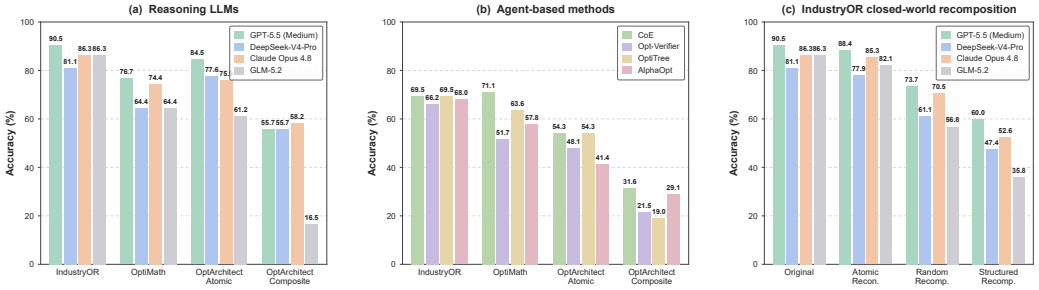


Figure 4: Main and closed-world results. Panels (a) and (b) reproduce the main benchmark comparison; panel (c) compares original IndustryOR problems with Atomic Reconstruction, Random Recomposition, and Structured Recomposition.

Table 5: Closed-world recomposition of IndustryOR mechanisms. Accuracy in %; Drop is the decrease from Atomic Reconstruction to Structured Recomposition.

Method	**Original**	**Atomic Recon.**	**Random Recomp.**	**Structured Recomp.**	**Drop**
GPT-5.5 (Medium)	90.5	88.4	73.7	60.0	28.4
DeepSeek-V4-Pro	81.1	77.9	61.1	47.4	30.5
Claude Opus 4.8	86.3	85.3	70.5	52.6	32.7
GLM-5.2	86.3	82.1	56.8	35.8	46.3
Average	**86.1**	**83.4**	**65.5**	**49.0**	**34.4**

Atomic Reconstruction remains close to the original benchmark, whereas both recomposition settings reduce accuracy. Structured Recomposition produces a 34.4-point average decrease from

Atomic Reconstruction, compared with 17.9 points for Random Recomposition. Since no external mechanism is introduced, the result isolates the effect of reorganizing mechanisms already present in IndustryOR.

D Analysis across Recursive Composition Depth

D.1 Accuracy across Recursive Depth

We define D_0 as problems generated from a single atomic skill, D_1 as problems produced by composing two skills, D_2 as problems whose construction involves at least one previously composed skill, and D_3 as deeper recursive compositions. Table 9 reports per-depth accuracy with 95% Wilson confidence intervals and the per-depth sample sizes. The aggregate results indicate that deeper compositions are more challenging, although the trend is not strictly monotonic for every model.

D.2 Structural Complexity across Recursive Depth

Table 6: Median structural statistics of problems at different recursive depths.

Depth	Variable Types	Constraint Families	Objective Terms	Solver Variables	Solver Constraints
D_0	3	4	1	40	40
D_1	5	6	1	109	94
D_2	8	9	4	262	290
D_3	14	12	6	651	909

The decrease in model accuracy is accompanied by a systematic increase in structural complexity. From D_0 to D_3, the number of decision-variable types increases from 3 to 14, constraint families from 4 to 12, and objective terms from 1 to 6. Solver-level model size also grows with depth.

D.3 Scale-Matched Complexity Analysis

To separate the effect of structural composition from the effect of problem size, we select, from the 116 Atomic problems, the subsets whose solver-level size falls within 30 percent of the average size of the composite problems at depth 1 and depth 2. The D1-matched subset contains 15 problems with on average 107 variables and 93 constraints, close to the D1 composite average of 109 variables and 94 constraints. The D2-matched subset contains 12 problems with on average 248 variables and 259 constraints, close to the D2 composite average of 262 variables and 290 constraints.

Table 7: Scale-matched comparison between Atomic and composite problems. Accuracy is averaged over the four reasoning models.

Setting	Avg. vars	Avg. constraints	Accuracy (%)
Atomic, D1-matched	107	93	80.0
Composite, depth 1	109	94	56.1
Atomic, D2-matched	248	259	85.4
Composite, depth 2	262	290	47.3

Table 7 reports the average accuracy of the four reasoning models on the matched subsets. At D1-matched size, Atomic problems are solved at 80.0 percent while composite problems at the same size are solved at 56.1 percent; at D2-matched size the corresponding numbers are 85.4 and 47.3 percent. Atomic accuracy does not decrease from the D1 band to the D2 band, whereas composed problems at matched size are solved 23.9 to 38.1 percentage points worse than Atomic problems. Problem size alone does not account for the gap: at matched size, composed problems remain 23.9–38.1 percentage points harder than Atomic problems, consistent with an additional effect from structural coupling.

E Statistical Reliability of Evaluation Results

Because the OptArchitect evaluation sets are smaller than conventional large-scale benchmarks, we additionally examine the statistical reliability of the reported results. We separately quantify the variation caused by stochastic decoding and the uncertainty arising from the finite number of evaluation problems. For the OptArchitect evaluation sets, we repeat inference over three independent decoding runs, report 95% confidence intervals for problem-level accuracy, and evaluate the sensitivity of the results to additional sampling through Pass@k.

We allocate more evaluation instances to lower recursive depths because deeper compositions require more costly construction and verification. The final evaluation set contains 116 Atomic problems and 79 Composite problems, with 37, 27, and 15 problems at D_1, D_2, and D_3. The enlarged allocation to deeper compositions reduces the instability caused by very small per-depth sample sizes while preserving broad coverage at D_1 and D_2.

Multiple-run stability. Table 8 reports accuracy over three independent decoding runs on the same evaluation problems. Accuracy varies across runs, but the magnitude of this variation is limited relative to the uncertainty associated with the finite evaluation sets. Run-to-run variation is smaller than the Atomic–Composite gap, so stochastic decoding does not account for the reported difference. Claude Opus 4.8 exhibits larger run-to-run variation on Atomic (std 6.0); its three-run mean nevertheless lies well within the reported confidence interval.

Table 8: Multiple-run stability on the OptArchitect evaluation sets. Each run uses the same evaluation problems and decoding configuration, with independent stochastic decoding at temperature 0.7; the standard deviation is computed across the three runs.

Method	Set	Run 1	Run 2	Run 3	Mean $\pm$ Std.
GPT-5.5 (Medium)	Atomic	81.9	87.1	84.5	84.5 ± 2.6
	Composite	53.2	57.0	57.0	55.7 ± 2.2
DeepSeek-V4-Pro	Atomic	75.9	79.3	77.6	77.6 ± 1.7
	Composite	53.2	58.2	55.7	55.7 ± 2.5
Claude Opus 4.8	Atomic	73.3	82.8	71.6	75.9 ± 6.0
	Composite	55.7	58.2	60.8	58.2 ± 2.6
GLM-5.2	Atomic	59.5	61.2	62.9	61.2 ± 1.7
	Composite	12.7	16.5	20.3	16.5 ± 3.8

Confidence intervals over the evaluation set. Table 9 reports 95% Wilson confidence intervals for problem-level accuracy at each recursive depth. The intervals become wider at deeper levels because fewer problems are available at D_2 and D_3. Nevertheless, the average accuracy decreases from 74.8% at D_0 to 21.7% at D_3. Individual models do not exhibit a strictly monotonic decline at every adjacent depth, as illustrated by the increase of DeepSeek-V4-Pro from D_1 to D_2. We therefore interpret the depth results as an aggregate difficulty pattern rather than as a strictly monotone law for every model.

Table 9: Recursive-depth accuracy (%) with 95% Wilson confidence intervals. The Average row reports the arithmetic mean across the four models and therefore has no binomial interval.

Method	D_0 (n=116)	D_1 (n=37)	D_2 (n=27)	D_3 (n=15)
GPT-5.5 (Medium)	84.5 [76.8, 90.0]	64.9 [48.8, 78.2]	55.6 [37.3, 72.4]	33.3 [15.2, 58.3]
DeepSeek-V4-Pro	77.6 [69.2, 84.2]	62.2 [46.1, 75.9]	66.7 [47.8, 81.4]	20.0 [7.0, 45.2]
Claude Opus 4.8	75.9 [67.3, 82.7]	70.3 [54.2, 82.5]	55.6 [37.3, 72.4]	33.3 [15.2, 58.3]
GLM-5.2	61.2 [52.1, 69.6]	27.0 [15.4, 43.0]	11.1 [3.9, 28.1]	0.0 [0.0, 20.4]
Average	74.8	56.1	47.3	21.7

Robustness to additional sampling. Table 10 evaluates whether the Atomic-to-Composite gap can be explained by decoding randomness. For each Composite problem, five candidate solutions are sampled using stochastic decoding with a temperature of 0.7, and Pass@k is estimated with

the standard unbiased estimator from the number of samples and the number of correct solutions. Additional sampling improves performance for all four models, but the improvement does not close the gap to Atomic accuracy, which indicates that decoding randomness contributes to performance variation but does not account for the observed difficulty.

Table 10: Robustness to additional sampling on OptArchitect-Composite.

Method	Pass@1	Pass@3	Pass@5
GPT-5.5 (Medium)	55.7	63.8	68.6
DeepSeek-V4-Pro	55.7	61.9	66.8
Claude Opus 4.8	58.2	65.4	70.1
GLM-5.2	16.5	24.7	30.8

F Synthesis Quality Evaluation

F.1 Automatic Synthesis Quality

Table 11: Synthesis quality of different generation strategies. All values are percentages; overall success is measured per transfer, with 100 transfers executed per strategy. Problem Correctness and Executable-Model Correctness are per-transfer synthesis-stage gate rates: they describe candidate yield rather than properties of the final accepted problems, all of which pass every gate of Appendix A.6.

Method	Problem Correctness	Executable-Model Correctness	Overall Success Rate
Direct Generation	6	15	7
Summary Prompting	11	27	12
OptArchitect-Atomic	82	88	65
OptArchitect-Composite	72	78	52

Table 11 reports a controlled comparison in which each strategy executes 100 transfers from the same corpora; the two generation baselines use the same generation backbone as OptArchitect (GLM-5.2) and differ only in how the source material is presented, as raw text or as a generated summary. All strategies receive the same transfer budget, the same regeneration budget after failed verification, and are graded by the same verifier under the same acceptance criteria (Section F.2), so the success-rate gap reflects the generation strategy rather than differences in budget or grading. Success is measured per transfer: a transfer is successful when one of its attempts produces an instance that passes all verification gates. Direct generation and summary-based prompting frequently produce ambiguous problems or inconsistent executable models. OptArchitect reaches higher problem-correctness and executable-model gate rates than either baseline. Composite synthesis has a lower yield than Atomic synthesis but remains well above direct and summary-based generation. These gate rates are candidate-level yields measured during synthesis; the accepted problems themselves pass all acceptance gates, and their residual quality as final problems is assessed by human experts in Appendix G.

F.2 Definition of a Successful Synthesis

A synthesized instance is counted as successful only if all of the following criteria hold: (i) the natural-language problem is internally consistent and specifies a well-defined optimization task; (ii) the executable reference model runs successfully and reproduces the certified answer; (iii) an independent reconstruction does not reveal a materially different defensible solution; (iv) the problem does not expose the target answer or intermediate solution through its description or data schema; and (v) the generated instance requires non-trivial modeling rather than reducing to direct lookup or a degenerate calculation.

G Human Expert Evaluation

G.1 Evaluation Protocol

Independent reviewers with expertise in optimization modeling assess a sampled subset of Atomic and Composite problems. Each problem is evaluated along five dimensions: problem correctness, modeling completeness, semantic clarity, practical plausibility, and solution accuracy. Each dimension is rated with a binary decision (yes or no).

We recruit three reviewers with operations research backgrounds, each with at least three years of experience in optimization modeling. Each reviewer evaluates 30 randomly sampled Atomic problems and 30 randomly sampled Composite problems, and the evaluation samples overlap across reviewers. Problems are presented in randomized order, and reviewers do not know how the problems were generated or whether a given problem is Atomic or Composite. Table 12 reports the percentage of yes ratings averaged across reviewers for each dimension.

G.2 Evaluation Criteria

Each problem is assessed along five dimensions. Problem correctness asks whether the problem is logically coherent and consistent with the provided data. Modeling completeness asks whether the required decision variables, objective, and constraints are sufficiently specified. Semantic clarity asks whether the problem statement admits a single unambiguous interpretation. Practical plausibility asks whether the decision context and the operational relationships it describes are plausible. Solution accuracy asks whether the stated question is answered by the certified reference solution. The first four dimensions are the primary targets of the evaluation; solution accuracy is auxiliary, because each reference solution is already certified by solver execution (Section 3.3), and the reviewers mainly check that the certified answer addresses the stated question.

G.3 Expert Evaluation Results

Table 12: Human expert evaluation of synthesized problems (%).

Method	Correctness	Completeness	Clarity	Plausibility	Solution Accuracy
OptArchitect-Atomic	90.0	100.0	95.0	98.0	96.6
OptArchitect-Composite	92.0	88.0	82.0	96.0	94.3

The expert evaluation confirms that the synthesized problems retain high correctness and practical plausibility. Composite problems exhibit lower modeling completeness and semantic clarity than Atomic problems, consistent with the additional difficulty of integrating multiple interacting modeling mechanisms.

G.4 Inter-Annotator Agreement

All dimensions use binary ratings, so agreement is computed over binary labels. On the problems evaluated by more than one reviewer, the average pairwise agreement across the five dimensions is 89.2%, and Fleiss' κ averaged over the five dimensions is 0.73, which indicates substantial agreement beyond chance.

H Ablation Studies

H.1 Effect of Structured Modeling Skills

Replacing the structured skill representation with a textual summary causes the largest degradation, reducing the success rate from 65% to 15% for Atomic synthesis and from 52% to 7% for Composite synthesis. This result suggests that summaries alone do not reliably preserve the modeling invariants, executable grounding, and transferable operations required for controlled synthesis.

Table 13: Ablation study measured by synthesis success rate (%).

Variant	Atomic	Composite
Full OptArchitect	65	52
w/o Structured Skill	15	7
w/o Matchmaker	30	22
w/o Bridge Operation	20	17
w/o Solver Verification	30	22
w/o Independent Probe	35	24
w/o Refinement	45	30

H.2 Effect of Compatibility Matching and Structural Coupling

Randomly selecting skill pairs reduces Composite synthesis success from 52% to 22%, while removing the structural bridge reduces it further to 17%. Both compatibility-based selection and explicit decision coupling materially affect composition reliability.

H.3 Effect of Verification

Removing solver verification, independent reconstruction, or skill-level refinement consistently reduces synthesis success. Solver verification detects invalid or inconsistent executable models, whereas the independent probe identifies semantic inconsistencies that cannot be detected from executability alone. Refinement further allows repeated failure patterns to be incorporated into subsequent generations.

I Cross-Backbone Generalization

I.1 Synthesis with Different Backbone Models

Table 14: Synthesis success rate (%) when applying the same OptArchitect skills with different generation backbones.

Backbone	Atomic	Composite
Qwen 3.8 Max	55	38
Hy 3	29	18
DeepSeek-V4-Pro	60	45
Kimi-K2.6	56	36

In all cross-backbone runs the reconstruction probe remains Gemini 3.5 Flash, so that the probe is always drawn from a different model family than the generator.

Synthesis success varies by backbone, but every evaluated generator can use the same skill representation.

I.2 Difficulty of Cross-Backbone Generated Problems

Table 15: Accuracy (%) of strong evaluator models on problems synthesized using alternative generation backbones.

Evaluator	Atomic	Composite
GPT-5.5 (Medium)	84.0	56.5
Claude Opus 4.8	75.5	64.5

We pool the accepted problems synthesized by the four alternative generation backbones and evaluate two strong reasoning models on the pooled set. The accuracies are close to the corresponding

results on the main OptArchitect evaluation sets in Table 1: GPT-5.5 (Medium) obtains 84.0% on Atomic and 56.5% on Composite, and Claude Opus 4.8 obtains 75.5% and 64.5%. Problems produced from the same skill library with different generation backbones have difficulty close to the main evaluation sets, so the observed gap is not specific to one generator.

J Efficiency and Computational Cost

Measurement setup. The reported token counts aggregate all generation and verification calls of the complete synthesis runs for each setting, and wall-clock times are averaged per problem, measured end to end from the start of Stage 1 to the acceptance or rejection of the final instance. Synthesis is bound by API calls to the generation backbone and by solver executions during verification.

J.1 Token Consumption

Table 16: Token consumption of OptArchitect synthesis.

Setting	Stage 1	Stage 2	Total
Atomic	3.52M	15.60M	19.12M
Composite	3.38M	18.22M	21.60M

Here Stage 1 denotes skill construction or recursive skill composition, while Stage 2 denotes the process from a verified skill to a final problem satisfying the synthesis criteria.

J.2 Wall-Clock Time

Table 17: Average wall-clock time per synthesized problem.

Setting	Stage 1	Stage 2	Total
Atomic	13 min	29 min	42 min
Composite	9 min	30 min	39 min

K Failure Analysis

K.1 Failure Taxonomy

We classify failure events collected from the single-skill and composite synthesis sessions, excluding API failures and framework bugs. Events are counted per attempt (Section 3.3), so a transfer that fails verification several times contributes several events. Each event receives exactly one primary label from the six categories below, and about 20 percent of the events also carry secondary labels. The distribution is cross-checked against the refinement notes recorded on the skill packages.

K.2 Recovery through Refinement

Detected failures are corrected inside the synthesis loop. Minor delivery issues are addressed by targeted regeneration of the problem statement, decorative couplings and data degeneration trigger regeneration of the full instance, and hard verification failures trigger the refinement loop, which updates the originating skill and regenerates the instance until it passes complete re-verification. Deep composites typically pass through multiple rounds of independent verification before the reference model and the reconstruction probe agree. Because refinement updates the reusable skill itself, its effect carries over to all later transfers from the same skill, whereas the other correction modes act on individual instances; Appendix L.1 shows a two-round example.

Table 18: Taxonomy of failure events. Percentages are computed over the failure events collected from the synthesis sessions; each event is assigned exactly one primary category, and about 20 percent of events also carry secondary labels.

Failure type (share %)	**Description**
Answer ambiguity (40)	The problem admits multiple defensible formulations or solutions, including temporal indexing ambiguity, under-specified functional forms or rate parameters, and alternative interpretations of mixed-integer constraints.
Formatting or information leakage (28)	The prompt or data exposes solution-relevant information, retains source-domain artifacts, directly names derived quantities, or fails delivery-format constraints.
Insufficient difficulty (10)	The instance can be solved through direct lookup, a single formula, or degenerates to a constant or trivial lower bound without requiring substantive modeling.
Structurally ineffective bridge (10)	A composite bridge changes the reported objective value but does not materially affect the optimal decision or feasible decision structure.
Solver implementation error (10)	The executable reference model contains an implementation error such as incorrect variable scope, double counting, or an incorrect mathematical transformation.
Other structural or data issues (2)	The skill representation lacks required operations or metadata, source provenance leaks into the output, or the provided data are insufficient to instantiate the mechanism.

L Case Studies of Verification and Refinement

L.1 Case Study I: Resolving Semantic Ambiguity

The first case concerns two closed-form transfer problems derived from the P11 PremadeFoods skill. The initial vaccine-vial-clinic and blood-bank-inventory realizations were rejected because the problem statements admitted multiple defensible interpretations.

The first source of ambiguity concerned the throughput construction: the description did not uniquely distinguish between a decaying pooled process and an interpretation based on the age of the most recently available unit. A second ambiguity concerned the composition of multiple value terms in the objective, where additive versus multiplicative aggregation and disposal-cost attribution were insufficiently specified.

The refinement mechanism converted these failures into reusable rules that require future realizations to specify the throughput construction and objective composition explicitly while avoiding disclosure of the final solution. After regeneration, independent verification identified two remaining issues: the qualitative behavior implied by the stated decay mechanism was inconsistent with the functional form used in the reference model, and the stochastic rate parameter could be interpreted as either a batch-level or unit-level arrival rate.

A second refinement therefore introduced rules requiring consistency between stated qualitative behavior and the implemented functional form, and requiring stochastic process rates to be tied explicitly to their generating process. The regenerated problems subsequently passed independent verification.

The two refinement rounds convert repeated ambiguity into reusable modeling guidance rather than an instance-specific repair.

L.2 Case Study II: Detecting an Executable but Incorrect Reference Model

The second case concerns a unit-commitment transfer to cold-chain compressor scheduling. The generated reference model executed successfully and returned an apparently valid optimum. However, independent reconstruction produced a different result.

Further diagnosis showed that the implementation incorrectly reused the per-unit initial state variable `x_prev` across unit-specific loops. As a result, the initial commitment state of one unit could silently inherit the state of another unit, relaxing the first-period ramp-up constraint and producing an infeasible but lower-cost solution.

The independent verifier detected the inconsistency, and a neutral adjudication step classified the failure as a reference-model implementation error rather than ambiguity in the natural-language problem. The resulting refinement introduced a general implementation rule requiring all unit-specific initial states to be recomputed or indexed explicitly within each per-unit modeling block.

L.3 Case Study III: Detecting Structurally Ineffective Composition

The third case concerns a recursively composed skill derived from P16, P06, and P01. Initial ablation results showed substantial changes in objective value when individual parent components were removed, apparently suggesting that all components contributed meaningfully to the composite model.

However, inspection of the optimal decisions revealed that removing two of the components changed only the objective accounting while leaving the selected facilities and served shares unchanged. The corresponding interactions therefore affected the reported score without materially changing the decision structure.

The bridge was therefore rejected under the structural-binding criterion of Equation equation 9, and the composite instance was regenerated with tighter shared capacity and resource competition. In the revised problem, removing each component changed the resulting optimal decision signature, confirming that the parent mechanisms were structurally coupled.

M Examples of Synthesized Problems

M.1 Atomic Example

This example is an OptArchitect-Atomic problem transferred from a surgical tray rationalization skill. The card summarizes the source mechanism, the target domain, the semantic mapping, and the verified result; Figure 5 illustrates the first three.

Source mechanism. A hospital decides how many instruments of each type each surgical tray retains (Figure 5a). Trimming is remove-only, a tray may not introduce instrument types it did not carry, and the plan must hit an overage-reduction target while keeping per-procedure-class shortage under a cap, so that a minimum number of trays require re-validation.

Target domain. Airline galley catering (Figure 5b). A regional carrier provisions each of 24 daily flights with 2 of 6 catering trolleys. Each trolley stocks a fixed integer count of each of 38 meal codes, and the seasonal manifests return 2185 uneaten portions per day against 987 served portions.

Semantic mapping. Tray maps to trolley, instrument type to meal code, procedure to flight, preference card to cabin class, point-of-use instrument usage to realized meal consumption, and instrument shortage to unserved passengers (Figure 5c). The structural invariant is preserved: a meal code may be retained in a trolley only if the trolley already stocks it, retained counts cannot exceed the status-quo manifest, and trolley-to-flight assignments are fixed.

Task and verified result. The problem asks for the minimum number of trolley manifests to republish so that total daily overage falls to at most 55 percent of the status quo while each of the 4 cabin classes keeps aggregate underage within 6 percent of its demand. The certified optimum modifies 3 of the 6 trolleys and reduces overage from 2185 to 1201.75 portions per day. The executable reference model contains 948 variables and 947 constraints.

(1) SOURCE PAPER FRAGMENT · OR source corpus, paper P28

Deshpande et al. (2024), *Operations Research* 72(5): 1874-1892, "Data-Driven Surgical Tray Optimization to Improve Operating Room Efficiency"

"…less than 20%-30% of reusable instruments supplied to a surgery are used on average… we formulate a data-driven optimization model for surgical tray configuration and assignment…"

tray rationalization stage, Problem (1):

$$
\begin{aligned}
V(T,\beta) = \min\ & \textstyle\sum_m \sum_i v_{m,i} && \text{(total overage)}\\
\text{s.t.}\quad & v_{m,i} \geq \textstyle\sum_{t \in T0(m)} x_{t,i} - u_{m,i} && \text{(supply – usage)}\\
& w_{m,i} \geq u_{m,i} - \textstyle\sum_{t \in T0(m)} x_{t,i} && \text{(shortage)}\\
& \textstyle\sum_{m \in S(p)} \sum_i w_{m,i} \leq (1-\beta) W^0_p \ \ \forall p && \text{(per-class service cap)}\\
& x_{t,i} \in \mathbb{Z}_+, \text{ remove-only, support fixed}
\end{aligned}
$$

source instantiation: facet = min total overage, baseline-relative cap

(a) The source mechanism.

(2) EXECUTABLE MODELING SKILL · $s = (O, I, \sigma, E)$ [Eq. (2)]

skill: kit-content-rationalization · deterministic MILP · difficulty 7.0

$\mathcal{O}$: 8 peer modeling operations; any non-empty subset yields a valid problem
`fixed-assignment rationalization` ..... mechanism (core)
`min total overage / min worst-case overage /`
`min total underage / min number-of-kits-modified` .. objective facets
counts direction: `remove-only` | `free-resize` ..... assumption variant
cap form: `baseline-relative` | `absolute` | `fraction-of--demand` variant
`per-class underage cap` ..... constraint

I : invariants:
- retain only already-held item types; counts ≤ status quo;
- `supply(job,item)` = ∑ retained counts over assigned kits;
- `overage` / underage = positive / negative part of (`supply` − `usage`)

σ : transfer signature (names no target domain): requires a fixed kit catalog with status-quo counts, jobs with observed kit assignments, realized point-of-use usage, job-clas partition; rejects re-optimized assignment, probability-of-need data, continuous item quantities

E : executable anchor: `data/*.csv` + `model.py` + `expected_solution.json`; deterministic, solver-certified by repeated execution ✓

(b) The target domain.

(3) SYNTHESIZED PROBLEM · OptArchitect-Atomic (D0) · airline galley catering

Northwind Galley Services provisions 24 daily flights; each galley loads exactly 2 of 6 trolleys (GAL1-6) stocking 38 meal codes; the seasonal manifests return 2185 uneaten portions/day against 987 served.

κ : semantic mapping: tray → trolley · instrument type → meal code · surgery → flight · preference card → cabin class · point-of-use usage → offload reconciliation · re-validation → manifest re-publication

ops facet = min number-of-kits-modified (≠ source facet) + remove-only + fraction-of-demand cap + overage-reduction target

data : packing manifest (60 rows) · trolley-flight assignment (48) · offload consumpion counts (441)

ASK : the minimum number of trolley manifests to re-publish so that total overage ≤ 55% of 2185 while each of the 4 cabin classes keeps its aggregate underage within 6% of its demand?

VERIFIED ✓ : solver-certified OPTIMUM: 3 of the 6 trolleys re-published; overage 2185 → 1201.75 portions/day; executable reference model with 948 variables / 947 constraints

(c) The semantic mapping.

Figure 5: The atomic example. (a) The source mechanism: rationalization of reusable surgical instrument trays. (b) The target domain: airline galley trolley meal-kit catering. (c) The semantic mapping from the source skill to the target domain.

M.2 Pairwise Composite Example

This example composes two atomic skills into a single problem; the result is one coupled decision problem, not a juxtaposition of two problems.

Parent skills. The first parent is a perishable-inventory skill: a product with a finite shelf life of L periods, per-period quality decay, and two coexisting age batches under a display rule, where a passive fraction of consumers takes the easiest unit first. The second parent is a joint-replenishment skill: multiple items share one ordering setup, and the common reorder cycle is restricted to a power-of-two multiple of a base interval so that items can share delivery epochs.

Structural bridge. The bridge couples the two mechanisms through the restock cycle R. The shared delivery setup forces R onto the power-of-two grid, while perishability requires $R < L \leq 2R$ so that at most two age lots coexist. Among the admissible grid values only one cycle length satisfies both conditions, and the order quantity, carryover, and disposal decisions are then optimized over this coupled cycle.

Target domain and verified result. The composed problem is instantiated as a vaccine cold-chain program. A clinic receives vaccine vials with a 12-week potency horizon on a synchronized refrigerated truck contract whose stop cadence is bound to a power-of-two grid of a two-week base interval. Two age lots may coexist in the refrigerator, and a passive fraction of nurses takes the easiest vial first. The task is to maximize the long-run average net benefit per week under administration fees, procurement, refrigeration, handling, disposal, biohazard wastage, and the shared truck setup cost. Among the four grid options only the 8-week cycle is admissible. The certified optimum is 52.40 dollars per week with 106 vials ordered per cycle, and an independent reconstruction reaches 52.46 dollars per week, a gap of 0.11 percent, which passes the dual-view verification.

M.3 Recursive Composite Example

This example composes three atomic skills with a previously verified depth-2 composite, so the resulting problem has recursive depth 3. The depth-2 parent is the composite described in Appendix L.3.

Parents. The primary layer composes a hub-location skill with a multiperiod capacitated planning skill: candidate sites are opened irreversibly over a three-year horizon, contracted throughput must be routed through opened site pairs, and last-mile delivery is capacitated with penalized shortage. A covering-fleet skill adds a scarce mobile-server layer: a small fleet of vans hosted at opened sites can serve nearby demand within a round-trip radius. The depth-2 parent contributes a choice-based facility layer with an affinity-gated roster: adoption follows a multinomial-choice model with a stay-at-home outside option, and a pairwise-rapport staff roster is gated by which sites are opened.

New bridges. Three couplings bind the parents into one decision structure. First, the van-served volume enters the per-zone charging balance, so the mobile fleet relieves grid capacity rather than forming a separate problem. Second, the effective service capacity of a station is derived from the number of maintenance shifts staffed there, so rostering and network design share one capacity variable. Third, the choice served-share of each zone is linked to the station-energization indicator through McCormick products, so the demand that the network must serve is itself created by the network decision.

Target domain and verified result. The composed problem is instantiated as a three-year roll-out of fast-charging stations for a regional electric-vehicle operator, with 4 candidate sites, 6 driver zones, a roaming battery-van fleet, and a technician roster. The certified optimum energizes 2 of the 4 stations and hosts van depots at both, at a net cost of 130.92. The executable model contains 353 variables and 699 constraints. A five-way ablation confirms that every layer is structurally binding: removing the roster layer, the van-balance term, the staff-derived capacity, or the choice-load coupling alters the optimal energization and dispatch decisions, and changes the optimal objective by between 30.7 and 832 percent relative to the full model.

N Difficulty-Controlled Problem Sequences

For a verified full problem, OptArchitect decomposes the formulation into modeling components and constructs a nested sequence by progressively restoring masked components. Each sequence satisfies structural monotonicity and independent executability.

$$\mathcal{L}^{(1)} \subset \mathcal{L}^{(2)} \subset \cdots \subset \mathcal{L}^{(K)}. \tag{15}$$

We require the first and final problems in a sequence to differ by the prescribed margin, and adjacent rounds to change the decision or objective. In our implementation, the relative difference between the seed and full problem is required to be at least 30%, while adjacent rounds are required to differ by at least 5%. Core structural mechanisms are introduced only after the initial round to avoid constructing a superficially simplified seed problem.

Table 19: A four-round difficulty sequence derived from the recursive example in Appendix M.3.

Round	Added mechanism	Vars	Constraints	Objective
1	Static energization with grid delivery and shortage penalty	62	74	64.2
2	Multiperiod irreversible energization and power-lane routing	154	198	85.5
3	Roaming battery-van covering layer	231	296	108.9
4	Logit adoption coupling and affinity-gated roster (full problem)	353	699	130.9

Table 19 shows a four-round sequence derived from the recursive example in Appendix M.3. Each round restores a set of masked modeling components and is independently verified. The relative difference between the seed and the full problem is 104 percent, and every adjacent pair differs by at least 20 percent, satisfying the construction thresholds. The core structural couplings enter only in rounds 3 and 4, so the seed problem does not leak the full modeling structure.

O Downstream Utility of Synthesized Problems

We also test OptArchitect as a source of training data. A 2K-example corpus combines existing optimization instructions with synthesized problems at multiple recursive depths, and is used to fine-tune Qwen3-8B for evaluation on six established benchmarks.

O.1 Training Setup

Following the data construction setting of ORLM (Huang et al., 2025), we use OR-Instruct as the seed corpus for downstream training. To obtain the application contexts used for synthesis, we cluster the OR-Instruct problems according to their topic keywords and use the resulting clusters to characterize representative optimization scenarios. OptArchitect then instantiates verified skills within these scenarios to generate additional training problems.

To avoid concentrating the synthesized data at a single level of structural complexity, we sample problems across different recursive depths. The resulting training corpus contains 2,000 problems in total, consisting of 1,000 examples sampled from OR-Instruct and 1,000 problems synthesized by OptArchitect. The synthesized portion contains both atomic and recursively composed problems, providing training examples with different levels of structural complexity. We initialize the model from Qwen3-8B and fine-tune it on the resulting mixture.

O.2 Evaluation Protocol

We evaluate the fine-tuned model on six optimization modeling benchmarks, including NL4OPT, MAMO EasyLP, MAMO ComplexLP, IndustryOR, ReSocratic, and OptiMath. We compare against ORLM, LLMOPT, OptMATH, and SIRL, which represent representative training-based approaches for optimization modeling. We report problem-level accuracy following the evaluation protocol of each benchmark.

O.3 Downstream Results

Table 20 compares the resulting model across six optimization-modeling benchmarks.

OptArchitect is best or tied best on five of the six benchmarks. Accuracy reaches 68.5% on MAMO ComplexLP, 51.0% on IndustryOR, and 70.1% on ReSocratic, compared with 61.1%, 48.0%, and 67.4% for SIRL. On OptiMath, OptArchitect reaches 44.6% versus 45.8% for SIRL.

Table 20: Downstream optimization modeling accuracy (%). Our model is based on Qwen3-8B and is fine-tuned on 1K OR-Instruct examples plus 1K OptArchitect-synthesized problems spanning different recursive depths. The best result in each column is shown in bold.

Method	NL4OPT	MAMO EasyLP	MAMO ComplexLP	IndustryOR	ReSocratic	OptiMath
ORLM	85.7	82.3	37.4	24.0	51.1	2.6
LLMOPT	80.3	89.5	44.1	29.0	53.8	12.5
OptMATH	95.9	89.9	54.1	31.0	66.1	34.7
SIRL	**98.0**	94.6	61.1	48.0	67.4	**45.8**
OptArchitect	**98.0**	**95.0**	**68.5**	**51.0**	**70.1**	44.6

The six-benchmark macro average is 71.2%, compared with 69.2% for SIRL. The largest gains occur on MAMO ComplexLP, IndustryOR, and ReSocratic, while NL4OPT and MAMO EasyLP remain near saturation. The result supports using recursively synthesized problems as additional supervision beyond the sampled OR-Instruct data.