RELIVE-OR: RETROSPECTIVE LEARNING THROUGH INTEGRATED VALIDATION OF EXPERIENCE FOR OPERATION RESEARCH MODELLING

Anonymous authors
Paper under double-blind review

ABSTRACT

Synthetic data is increasingly used to fine-tune large language models (LLMs) for operations research (OR) modeling, yet scaling synthesis alone does not necessarily produce better OR modelers. We view each problem–program pair as a *modeling experience* and OR training as a process of acquiring, validating, revisiting, and reusing such experiences. This perspective exposes a key limitation of existing synthesis pipelines: beyond semantic, code-structural, and outcome consistency, a synthetic experience should remain consistent with the *OR model specification* represented by its trusted seed. Otherwise, it may remain executable and numerically valid while becoming misspecified with respect to the intended modeling principle. We propose **RE**trospective **L**earning through **I**ntegrated **V**alidation of **E**xperience (**RELIVE-OR**) for OR modeling—with the objective of maximizing structural freedom while preserving OR model specification. Starting from a small trusted seed set, RELIVE-OR synthesizes structurally diverse experiences directly in executable code space and applies closed-loop verification and repair for semantic, code-structural, and outcome consistency. The verified experiences are first used for broad consolidation to expose the learner to diverse OR structures. RELIVE-OR then retrospectively estimates the influence of each verified experience on a trusted seed objective and re-selects experiences that remain aligned with the intended OR model specification for focused consolidation. Finally, the verified experience bank is reused at test time as a non-parametric reference memory for gradient-guided selection among executable candidates, without additional parameter updates. Across six established OR modeling benchmarks, RELIVE-OR enables an 8B model to achieve state-of-the-art or competitive performance and perform on par with substantially larger frontier models.

1 INTRODUCTION

Synthetic data is increasingly used to fine-tune large language models (LLMs) for operations research (OR) modeling, where models translate natural-language decision problems into mathematical formulations and solver-executable programs (Ramamonjison et al., 2023; AhmadiTeshnizi et al., 2024). Recent work has expanded synthetic OR data through scenario and formulation augmentation, reverse generation, template-based synthesis, and direct evolution of optimization programs (Huang et al., 2025b; Yang et al., 2025; Lu et al., 2025; Li et al., 2025; Motwani et al., 2026). Complementary approaches improve optimization modeling through progressive difficulty generation, solver-feedback reinforcement learning, and structured reasoning (Wu et al., 2025; Chen et al., 2026; Xiao et al., 2026).

Nevertheless, a key challenge persists: useful synthesis requires *structural freedom* to expand beyond the limited structures represented by trusted data, yet greater freedom also increases the risk of violating the intended OR model specification. Restricting synthesis too strongly produces redundant variants, whereas unconstrained synthesis may alter variable roles, constraint logic, or objective semantics. The central challenge is therefore to *maximize structural exploration while preserving the underlying OR model specification (Sec. 2.2)*. Inspired by experiential and reflective learning (Kolb, 1984), we take an orthogonal perspective: we view each problem–program pair as a *modeling experience*, and OR training as a process of acquiring, validating, revisiting, and reusing such experiences. This perspective reflects the structural nature of OR expertise: recurring model-

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.

ing principles—such as binary decisions, logical implications, Big-M formulations, indicator constraints, piecewise-linear functions, and auxiliary variables—are repeatedly adapted across diverse decision problems rather than memorized as isolated formulations. This perspective also exposes a limitation of existing synthesis pipelines. Beyond semantic and formulation consistency (Yang et al., 2025; Lu et al., 2025), code executability and repair (AhmadiTeshnizi et al., 2024), and solver- or outcome-based verification (Lu et al., 2025; Chen et al., 2026; Motwani et al., 2026), a synthetic experience should remain consistent with the OR model specification represented by its originating trusted seed. Yet trusted seeds provide only partial structural coverage of the unknown target OR distribution. The challenge is therefore to expand structural coverage efficiently under a finite synthesis budget without allowing useful structural variation to become *OR model misspecification*.

To address this challenge, we introduce **RELIVE-OR**, an experience-centric framework for synthesizing, verifying, revisiting, and reusing OR modeling experiences. RELIVE-OR is built around the principle of *maximizing structural freedom under OR model specification preservation*. Its **DE-SCALE** component (Sec. 2.1) treats solver-executable programs as *executable symbolic latents* and directly transforms variables, domains, objectives, constraints, and their relationships to generate structurally diverse experiences. The transformed program then serves as an anchor in a code–text–code loop that verifies semantic, code-structural, and outcome consistency through reconstruction, execution, and repair. ge These checks establish whether a synthesized experience is technically reliable, but they do not by themselves ensure consistency with the broader OR model specification represented by the trusted seeds. Moreover, even a verified pool may remain redundant, imbalanced, or structurally concentrated after broad learning. RELIVE-OR therefore first performs *broad consolidation* over the verified experiences and then retrospectively re-evaluates them by their estimated influence on a trusted seed objective (Sec. 2.3). Experience validity asks whether a synthesized experience is reliable, whereas retrospective utility asks whether it remains useful to the current learner and aligned with trusted OR behavior. High-utility experiences are selected for focused consolidation, improving consistency with trusted OR specifications. We further reuse the verified experience bank at inference time through **ReFerence-Guided UpdatE-Free Test-Time Program Selection (FREE)**, which ranks executable candidates without parameter updates (Sec. 2.4). Our main contributions are:

- **A New Perspective:** Unlike prior work that largely treats synthetic OR data as a static collection of training examples, we view each problem–code pair as a *modeling experience* and OR learning as a process of acquiring, validating, revisiting, and reusing such experiences. This perspective motivates the central objective of *expanding structural coverage while preserving seed-derived OR model specifications*.
- **A New Framework:** We propose **RELIVE-OR**, which integrates executable-space structural synthesis, closed-loop verification, retrospective re-selection, and update-free test-time reuse. **DE-SCALE** diversifies executable formulations, retrospective influence selects useful verified experiences, and **FREE** reuses them for parameter-free test-time program selection.
- **Strong Empirical Performance:** Across six established OR modeling benchmarks, RELIVE-OR substantially improves the performance of an 8B model and achieves competitive results relative to substantially larger general-purpose models. The results further show that retrospective experience selection and update-free reference-guided selection provide complementary gains beyond broad synthetic-data training.

2 RETROSPECTIVE LEARNING THROUGH INTEGRATED VALIDATION OF EXPERIENCE

Problem Setting Let $\mathcal{D}_0 = \{(x_i^0, z_i^0)\}_{i=1}^{N_0}$ denote a validated seed set, where x_i^0 is a natural-language OR problem, z_i^0 is its solver-executable formulation code, and N_0 is the number of seed pairs. We regard $\mathcal{D}_0$ as a small trusted sample from an unknown target OR distribution $P^\star$, i.e., $(x_i^0, z_i^0) \sim P^\star$. Ideally, an OR model would be trained on a large and structurally diverse collection of verified problem–code pairs. In practice, such high-quality data are scarce, and $\mathcal{D}_0$ alone is insufficient for effective training. A natural solution is therefore to expand the seed set through synthetic data generation. However, increasing data quantity alone does not necessarily increase diversity in the underlying optimization structures and their code realizations. To improve structural coverage, starting from a trusted seed $(x_i^0, z_i^0) \in \mathcal{D}_0$, we perturb the executable formulation-

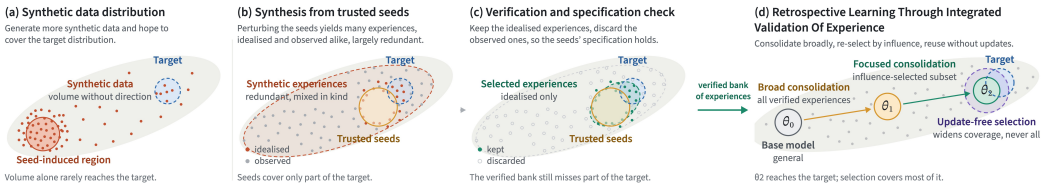

Figure 1: **RELIVE-OR: from diverse synthesis to retrospective learning.** Trusted-seed perturbation expands structural coverage, while integrated validation removes off-specification experiences and yields the verified bank $\mathcal{D}_V$. Broad consolidation produces θ_1, retrospective influence and focused consolidation produce θ_2, and update-free selection reuses verified experiences at inference time without parameter updates.

code space and construct a larger synthetic pool $\mathcal{D}_{\text{syn}} = \{(x'_j, z'_j)\}_{j=1}^{N}$, where $(x'_j, z'_j) \sim Q_{\text{syn}}$ and $N \gg N_0$. Here, Q_{syn} denotes the distribution induced by the synthesis process. While such perturbations increase structural diversity, synthesis may also introduce systematic deviations from the target distribution $P^\star$ through structural imbalance, structural drift, mode concentration, incomplete coverage, or other synthesis artifacts. We refer to this distributional mismatch as *synthetic-data misspecification*. Consequently, a model trained directly on the synthetic distribution, $\hat{\theta}_{\text{syn}} = \arg\min_\theta \mathbb{E}_{(x,z)\sim Q_{\text{syn}}}\big[\ell(f_\theta(x), z)\big]$, may become misspecified with respect to the target OR distribution. Our goal is therefore to retain the structural diversity introduced by synthesis while selecting synthetic problem–code pairs whose training effects remain aligned with the trusted OR behavior anchored by $\mathcal{D}_0$.

2.1 Stage I: Broad Experience Acquisition

The first stage equips the model with broad OR modeling experience by training on natural-language problems paired with solver-executable formulation code, where each verified pair represents an optimization specification and its executable realization. However, high-quality natural-language–code pairs remain scarce. Existing approaches expand training data through scenario augmentation, reverse generation, and generator-based synthesis Huang et al. (2025a); Lu et al. (2025), but scaling synthetic data can yield diminishing returns when generated examples repeatedly instantiate similar optimization structures. We therefore ask: *can synthetic OR experience be diversified directly in executable optimization space?* Building on this perspective, we propose ***D**ata-**E**fficient **S**cenario-**C**onditioned Structur**A**l **L**atent Synth**E**sis* (***DE-SCALE***) as the experience-generation component of RELIVE-OR.

2.1.1 Executable Programs as Symbolic Latents

RELIVE-OR introduces diversity directly in executable OR program space and verifies whether the resulting decision structure can be faithfully expressed and recovered through natural language. The synthesis process is $(x_0, z_0) \to z' \to x' \to \hat{z}' \to \text{verify} \to \text{repair/rewrite/filter} \to \mathcal{D}_{\text{train}}$. Because natural-language descriptions may vary substantially and leave modeling structure implicit, we operate on solver-executable formulation code, which explicitly encodes decision variables, objectives, constraints, parameters, and domains. Let $x \in \mathcal{X}$ denote a natural-language OR problem and $z \in \mathcal{Z}_{\text{OR}}$ its solver-executable formulation code. We represent $z = (\mathcal{V}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{B})$ and view it as an *executable symbolic latent* of x: discrete, interpretable, directly executable, and largely invariant to linguistic variation.

2.1.2 Scenario-Conditioned Structural Latent Synthesis

Given a validated seed program z_0, a scenario s specifies a new decision setting, while a structural scaffold h provides generic OR modeling guidance. We denote the scenario-conditioned code synthesis operator by $\mathcal{T}_\phi$, where ϕ denotes the parameters of the LLM used for synthesis. The synthesized executable program is $z' \sim \mathcal{T}_\phi(z \mid z_0, s, h)$. Unlike instance-level parameter perturbation, this transformation may modify variable families, variable domains, objective functions, constraints, indexing structures, and their structural relationships. The resulting program z' therefore represents

a new executable optimization experience derived from the modeling principles represented by the seed and structural scaffold. Importantly, z' is not only a generated training target. It also serves as the executable anchor against which the corresponding natural-language experience and its reconstructed formulation code are validated.

2.1.3 Integrated Validation of Synthesized Experience

Structural diversity is useful only when the synthesized experience remains semantically and structurally reliable. RELIVE-OR therefore performs the first level of integrated validation during experience construction through DE-SCALE. Using the same synthesis LLM with different prompting directions, we first verbalize the synthesized executable program z' into a natural-language problem x' and then independently reconstruct solver-executable formulation code $\hat{z}'$: $x' \sim p_\phi(x \mid z')$, and $\hat{z}' \sim p_\phi(z \mid x')$, yielding the closed loop $z' \to x' \to \hat{z}'$. Here, p_ϕ denotes the conditional generation distribution of the shared synthesis LLM under code-to-text and text-to-code prompting, respectively. Exact source-code equality between z' and $\hat{z}'$ is not required. Instead, let $S(z', \hat{z}') \in \{0, 1\}$ denote structural consistency between the synthesized and reconstructed formulation code. The verifier evaluates optimization-specific structural properties, including objective direction, variable domains, and source-dependent OR structures such as flow or inventory balance, conic constraints, subtour-elimination constraints, startup and ramping logic, and sequencing constraints. Let $E(z) \in \{0, 1\}$ indicate that formulation code z executes successfully and returns a finite objective, and let $f(z)$ denote the corresponding objective value. We define numerical agreement by

$$|f(z') - f(\hat{z}')| \le \max\big(10^{-6}, 10^{-3} \max\{|f(z')|, |f(\hat{z}')|\}\big) .$$

The single-view executable verification criterion is

$$V(z', \hat{z}') = \mathbb{I}[S(z', \hat{z}') = 1 \wedge E(z') = 1 \wedge E(\hat{z}') = 1 \wedge f(\hat{z}') \simeq f(z')] ,$$

where $\mathbb{I}[\cdot]$ is the indicator function and $f(\hat{z}') \simeq f(z')$ denotes the numerical agreement condition above. This verification loop provides a discrete form of *structural denoising*. Rather than removing continuous noise, the system detects and corrects symbolic inconsistencies such as incorrect variable domains, missing constraints, invalid solver logic, or incomplete natural-language specifications.

2.1.4 Multi-View Verification and Repair

If x' is malformed, structurally incomplete, or numerically inconsistent, we rewrite it while keeping z' fixed as the executable anchor. This prevents an unreliable textual realization from unnecessarily altering the synthesized optimization structure. The importance of this step is examined in Section E. To reduce rejection caused by a single lossy verbalization, we generate up to $K_{\text{view}} = 5$ independent textual views: $x'_k \sim p_\phi(x \mid z')$, $\hat{z}'_k \sim p_\phi(z \mid x'_k)$, $k \in [K_{\text{view}}]$, where $[K_{\text{view}}] = \{1, \ldots, K_{\text{view}}\}$. The synthesized program passes multi-view executable verification if at least one view is successfully reconstructed:

$$V_{\text{view}}(z') = \mathbb{I}[\exists k \in [K_{\text{view}}] : V(z', \hat{z}'_k) = 1] .$$

If $\hat{z}'_k$ fails structural validation, execution, or finite-objective extraction, verifier feedback is used for bounded code repair, followed by revalidation and execution. In contrast, if $\hat{z}'_k$ executes successfully but its finite objective disagrees with z', the reconstructed code is not repeatedly modified by default. Instead, the corresponding textual realization is discarded and regenerated from the unchanged executable anchor. This distinction separates failures of executable reconstruction from failures of natural-language realization and preserves the synthesized optimization structure whenever possible.

2.1.5 Verified Experience Pool and Broad Consolidation

The synthesis stage produces a large pool of candidate problem–code pairs drawn from Q_{syn}. Let j index a synthesized candidate $r_j = (x'_j, z'_j)$ derived from the seed pair $r^0_{\pi(j)} = (x^0_{\pi(j)}, z^0_{\pi(j)}) \in \mathcal{D}_0$. Not every synthesized candidate is retained. In addition to the multi-view executable verification above, we require the synthesized experience to preserve the seed-derived OR specification associated with its originating seed. We therefore define the final validation indicator as $V_{\text{final}}\left(x'_j, z'_j; x^0_{\pi(j)}, z^0_{\pi(j)}\right) \in \{0, 1\}$, where acceptance requires the synthesized experience to satisfy semantic, code-structural, execution, outcome, and seed-relative model-specification checks[1].

[1] See Sec. D for structural-diversity evidence.

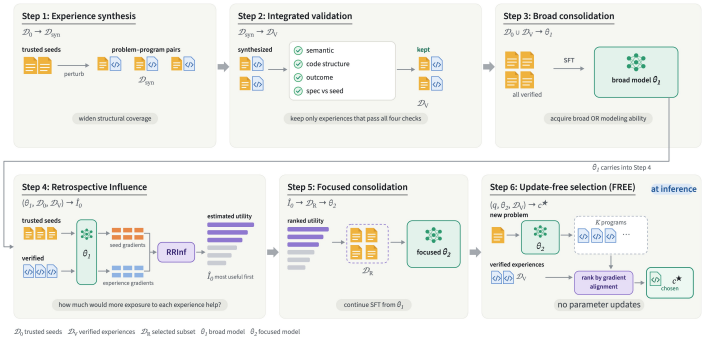

Figure 2: **RELIVE-OR** treats OR modeling as experience learning: it synthesizes and verifies diverse executable experiences, consolidates them broadly, retrospectively re-selects the most useful experiences, and reuses verified experiences for update-free test-time program selection.

The verified synthetic experience pool is therefore

$$\mathcal{D}_{\mathrm{V}} = \left\{ (x'_j, z'_j) \in \mathcal{D}_{\mathrm{syn}} : V_{\mathrm{final}} \left(x'_j, z'_j; x^0_{\pi(j)}, z^0_{\pi(j)} \right) = 1 \right\}. \tag{1}$$

The broad training set combines the trusted seeds with these verified synthetic experiences as $\mathcal{D}_{\mathrm{train}} = \mathcal{D}_0 \cup \mathcal{D}_{\mathrm{V}}$. Executable formulation code therefore plays three complementary roles in RELIVE-OR: it provides the space in which structural diversity is introduced, serves as an executable anchor during semantic and specification verification, and supplies supervision for downstream OR modeling. The learner is first adapted on the complete verified training set, $\theta_1 = \arg\min_\theta \frac{1}{|\mathcal{D}_{\mathrm{train}}|} \sum_{(x,z)\in\mathcal{D}_{\mathrm{train}}} \ell(f_\theta(x), z)$. This stage emphasizes *broad experience acquisition*, exposing the model to diverse realizations of recurring OR modeling principles before retrospective selection. Verification ensures that each retained synthetic pair is technically valid and consistent with the seed-derived OR specification of its originating seed, but it does not guarantee that the verified pool as a whole forms the most useful training distribution for the current learner. In particular, individually specification-consistent experiences may collectively overrepresent certain objective, constraint, or formulation structures while under-covering others. Hence, pair-level specification consistency $\not\Rightarrow$ dataset-level training utility.This motivates the next stage of RELIVE-OR: after broad consolidation to θ_1, we retrospectively identify which already-valid experiences are predicted to most improve the trusted reference objective for the *current learner*.

2.2 OR Model Specification and Misspecification

Inspired by the model-misspecification perspective of Pan et al. (2022), we view synthesized OR experiences as observed data whose generating distribution may violate the intended OR specification. Let $r = (x, z)$ denote an OR modeling experience, where x is a natural-language OR problem and z is its solver-executable formulation code. Let $P^\star$ denote the unknown target distribution of desirable OR experiences.

Finite-budget trusted-conditioned synthesis. Under a limited synthesis budget, unconditional generation from Q^{u}_ϕ has no target-specific anchor and therefore provides no guarantee of either specification validity or sufficient target coverage. We instead assume access to a small validated seed set $\mathcal{D}_0 = \{r^0_i\}_{i=1}^{N_0}, r^0_i = (x^0_i, z^0_i)$, which is correctly specified and target-relevant, but need not be an i.i.d. sample from $P^\star$. Its limited size and structural coverage, however, make direct training insufficient. We therefore condition stochastic LLM-based synthesis on the trusted seeds, new scenarios s_j, and structural scaffolds h_j:

$$r_j = (x'_j, z'_j) \sim Q^{\mathrm{c}}_\phi \left(\cdot \mid r^0_{\pi(j)}, s_j, h_j \right), \qquad \mathcal{D}_{\mathrm{syn}} = \{r_j\}_{j=1}^{N}. \tag{2}$$

Marginalizing over seed selection, scenarios, scaffolds, and stochastic generation induces the synthesis distribution Q_{syn}. The pretrained knowledge of the LLM and scenario conditioning allow synthesis to expand beyond structures explicitly represented by $\mathcal{D}_0$, while the trusted seeds provide target-relevant modeling guidance. Trusted conditioning therefore reduces uncontrolled synthesis, but does not eliminate misspecification.

OR model specification and misspecification. For an OR problem x, let $\mathrm{Spec}(x)$ denote the problem-specific requirements that a valid formulation code z should satisfy, including decision semantics, variable domains, objective intent, constraint roles, logical relationships, and other user- or client-specified requirements. Define the correctly specified OR experience space as

$$\mathcal{R}_{\mathrm{OR}} = \{(x, z) : z \text{ satisfies } \mathrm{Spec}(x)\}, \tag{3}$$

and the corresponding admissible family as

$$\mathcal{P}_{\mathrm{OR}} = \{P : \mathrm{supp}(P) \subseteq \mathcal{R}_{\mathrm{OR}}\}, \qquad P^\star \in \mathcal{P}_{\mathrm{OR}}. \tag{4}$$

The term *OR model misspecification* refers to the mismatch between the admissible OR family $\mathcal{P}_{\mathrm{OR}}$ and the observed synthesis distribution Q_{syn}, namely synthesized problem–code pairs do not strictly satisfy the intended problem-specific OR requirements. Formally,

$$P^\star \in \mathcal{P}_{\mathrm{OR}}, \qquad Q_{\mathrm{syn}} \notin \mathcal{P}_{\mathrm{OR}}, \qquad r_j \sim Q_{\mathrm{syn}}. \tag{5}$$

Thus, a synthesized formulation may be executable and numerically valid while still being misspecified if it violates the intended OR specification. Specification validity is distinct from target-distribution alignment. Even if synthesis produces only specification-valid experiences,

$$Q_{\mathrm{syn}} \in \mathcal{P}_{\mathrm{OR}} \quad \not\Rightarrow \quad Q_{\mathrm{syn}} \approx P^\star, \tag{6}$$

because finite-budget synthesis may remain redundant, structurally imbalanced, mode-concentrated, or incomplete.

Partial structural coverage and verification. Let $\psi(r)$ denote the structural attributes of an OR experience and define

$$\mathcal{C}_0 = \{\psi(r_i^0) : r_i^0 \in \mathcal{D}_0\}, \qquad \mathcal{C}^\star = \mathrm{supp}(\psi_\# P^\star). \tag{7}$$

We assume $\mathcal{C}_0 \subsetneq \mathcal{C}^\star$, so useful synthesis must expand beyond the limited structures represented by the trusted seeds.

For each synthetic experience r_j, let $\pi(j)$ denote its originating seed and define

$$\Gamma_j = \mathrm{Spec}_{\mathrm{syn}}\left(r^0_{\pi(j)}, s_j, h_j\right) \tag{8}$$

as the intended specification induced by the seed, scenario, and structural scaffold. Let $\mathcal{F}_{\mathrm{spec}}(\Gamma_j)$ denote its admissible realizations. We retain r_j only if

$$r_j \in \mathcal{F}_{\mathrm{spec}}(\Gamma_j). \tag{9}$$

Hence, structural variation is permitted as long as the intended specification is preserved. Integrated validation removes identifiable pair-level specification violations and yields the verified pool $\mathcal{D}_{\mathrm{V}}$.

2.3 Retrospective Experience Re-selection

Let Q_{V} denote the distribution induced by the verified experience pool $\mathcal{D}_{\mathrm{V}}$. Seed conditioning and verification constrain synthesis, but they do not guarantee that the verified pool is well aligned with the target distribution. In particular,

$$r_j \in \mathcal{F}_{\mathrm{spec}}(\Gamma_j) \quad \forall\, r_j \in \mathcal{D}_{\mathrm{V}} \quad \not\Rightarrow \quad Q_{\mathrm{V}} \approx P^\star. \tag{10}$$

Thus, even specification-valid synthesis may remain redundant, mode-concentrated, structurally imbalanced, or unevenly distributed over valid OR structures. This residual mismatch motivates retrospective re-selection rather than treating all verified experiences as equally useful. Since $P^\star$ and the test set are unavailable during training, $\mathcal{D}_0$ serves as a finite trusted reference for desirable OR behavior. After broad consolidation to θ_1, we therefore evaluate each verified experience by its local effect on the trusted reference objective. Retrospective influence prioritizes experiences whose additional exposure is predicted to improve this objective, thereby mitigating residual training misalignment beyond seed-conditioned synthesis and verification.

From misspecification to retrospective influence. Verification removes identifiable specification violations, but the verified distribution Q_{V} may still differ from the unknown target distribution $P^\star$ due to redundancy, imbalance, or incomplete structural coverage. Since $P^\star$ is inaccessible, RELIVE-OR instead uses the trusted reference set $\mathcal{D}_0$ to assess which verified experiences remain useful after broad consolidation to θ_1. Specifically, influence estimates how upweighting a verified experience would change the loss on $\mathcal{D}_0$. Thus, retrospective influence re-selects among already-valid experiences rather than serving as another verification test. Recall that $\mathcal{D}_{\mathrm{train}} = \mathcal{D}_0 \cup \mathcal{D}_{\mathrm{V}}$, and let θ_1 denote the model obtained by broad consolidation on $\mathcal{D}_{\mathrm{train}}$.

Trusted reference objective. Because the target OR distribution $P^\star$ is unavailable, we use the trusted reference set $\mathcal{D}_0 = \{r_i^0 = (x_i^0, z_i^0)\}_{i=1}^{N_0}$ as an empirical anchor for desirable OR behavior. We define

$$\mathcal{L}_0(\theta) = \frac{1}{N_0} \sum_{i=1}^{N_0} \ell(f_\theta(x_i^0), z_i^0). \tag{11}$$

Although $\mathcal{D}_0$ participates in broad consolidation, here it serves as a trusted reference for evaluating whether additional exposure to a verified experience would improve the current learner.

Retrospective influence. Let $r_j = (x_j', z_j') \in \mathcal{D}_{\mathrm{V}}$ be a verified synthetic experience, and define

$$\ell_j(\theta) = \ell(f_\theta(x_j'), z_j'), \qquad g_j = \nabla_\theta \ell_j(\theta_1), \qquad g_0 = \nabla_\theta \mathcal{L}_0(\theta_1). \tag{12}$$

Here, g_j is the gradient induced by the candidate experience r_j, while g_0 is the gradient of the trusted reference objective. To quantify the effect of additional exposure to r_j, consider its infinitesimal upweighting:

$$\theta_{\epsilon,j} = \arg\min_\theta \left[R_{\mathrm{train}}(\theta) + \epsilon\, \ell_j(\theta) \right], \tag{13}$$

where

$$R_{\mathrm{train}}(\theta) = \frac{1}{|\mathcal{D}_{\mathrm{train}}|} \sum_{r=(x,z)\in\mathcal{D}_{\mathrm{train}}} \ell(f_\theta(x), z). \tag{14}$$

The retrospective influence of r_j on the trusted reference objective is $I_0(r_j) = \left.\frac{\partial \mathcal{L}_0(\theta_{\epsilon,j})}{\partial \epsilon}\right|_{\epsilon=0}$. Under the standard local influence approximation Koh & Liang (2017); Pruthi et al. (2020) around θ_1, $I_0(r_j) = -g_0^\top H_1^{-1} g_j, H_1 = \nabla_\theta^2 R_{\mathrm{train}}(\theta_1)$. Here, g_0 provides the trusted reference direction, H_1 captures the local curvature of the broadly consolidated model, and g_j represents the candidate experience being evaluated. Under this sign convention, $I_0(r_j) < 0$ means that increasing the weight of r_j is predicted to decrease the trusted reference loss. Thus, more negative influence indicates greater retrospective utility. For the negative log-likelihood objective used in language-model training, RRInf (Tu et al., 2025) replaces the Hessian by the empirical second moment of per-example gradients and applies damping. Let $n = |\mathcal{D}_{\mathrm{train}}|$, enumerate $\mathcal{D}_{\mathrm{train}} = \{u_k\}_{k=1}^n$, and define $g_k = \nabla_\theta \ell(u_k; \theta_1), G_1 = \frac{1}{n}\sum_{k=1}^n g_k g_k^\top$. The corresponding damped influence target is

$$\widetilde{I}_0(r_j) = -g_0^\top (G_1 + \lambda I_d)^{-1} g_j, \qquad \lambda > 0. \tag{15}$$

Here, I_d is the d-dimensional identity matrix. In this formulation, g_0 is computed from the trusted reference set $\mathcal{D}_0$, G_1 captures the curvature induced by the broad-training set $\mathcal{D}_{\mathrm{train}}$, and g_j corresponds to the verified experience being evaluated. influence values are computed without explicitly forming the inverse. Let $\Phi = [g_1, \ldots, g_n] \in \mathbb{R}^{d\times n}$ and $\upsilon = -n g_0 \in \mathbb{R}^d$. Collecting the damped influences of all training experiences into $\widetilde{\mathbf{I}} \in \mathbb{R}^n$, which gives

$$\widetilde{\mathbf{I}} = \arg\min_{\boldsymbol{\omega}\in\mathbb{R}^n} \frac{1}{d} \|\Phi\boldsymbol{\omega} - \upsilon\|_2^2 + \widetilde{\lambda}\|\boldsymbol{\omega}\|_2^2, \qquad \widetilde{\lambda} = \frac{n\lambda}{d}. \tag{16}$$

In practice, RRInf solves this ridge problem efficiently using its stochastic estimator; we denote the resulting numerical influence estimate for $r_j \in \mathcal{D}_{\mathrm{V}}$ by $\widehat{I}_0(r_j)$. RELIVE-OR retains only the components corresponding to verified synthetic experiences for retrospective valuation.

Retrospective re-selection. Given a fixed retrospective budget B, we select the verified experiences with the smallest estimated influence values:

$$\mathcal{D}_{\mathrm{R}} = \arg\min_{\substack{\mathcal{A}\subseteq\mathcal{D}_{\mathrm{V}} \\ |\mathcal{A}|=B}} \sum_{r_j\in\mathcal{A}} \widehat{I}_0(r_j). \tag{17}$$

Equivalently, let $S_{\mathrm{retro}}(r_j) = -\widehat{I}_0(r_j)$ and retain the top-B verified experiences, with $B = 0.5|\mathcal{D}_{\mathrm{V}}|$. These are predicted to yield the largest reduction in the trusted reference loss. For any $\mathcal{A} \subseteq \mathcal{D}_{\mathrm{V}}$ with $|\mathcal{A}| = B$,

$$\theta_{\epsilon,\mathcal{A}} = \arg\min_\theta \left[R_{\mathrm{train}}(\theta) + \frac{\epsilon}{B} \sum_{r_j\in\mathcal{A}} \ell_j(\theta) \right], \tag{18}$$

and $\mathcal{L}_0(\theta_{\epsilon,\mathcal{A}}) = \mathcal{L}_0(\theta_1) + \frac{\epsilon}{B}\sum_{r_j\in\mathcal{A}} I_0(r_j) + o(\epsilon)$. Thus, selecting the B smallest influence values gives the fixed-size subset with the greatest predicted first-order reduction in the trusted reference objective. We then perform focused consolidation from θ_1: $\theta_2 = \mathrm{SFT}(\theta_1, \mathcal{D}_{\mathrm{R}})$.

2.4 REFERENCE-GUIDED UPDATE-FREE TEST-TIME PROGRAM SELECTION (**FREE**)

The verified experience bank is further exploited at test time without parameter updates. This mirrors retrospective influence during training: whereas retrospective re-selection evaluates verified experiences against the trusted seed objective, test-time selection evaluates *candidate programs* against retrieved verified experiences using a related curvature-aware gradient geometry. Given an unseen OR problem q, the fixed model f_{θ_2} generates $\mathcal{C}(q) = \{c_1, \ldots, c_{K_{\text{gen}}}\}$. We execute each candidate and retain $\mathcal{E}(q) = \{c \in \mathcal{C}(q) : \text{Exec}(c) \text{ returns a finite numeric objective}\}$, where execution serves only as an eligibility check and is not used to rank the retained candidates.

Verified reference retrieval. Let $\mathcal{B} = \{(q_i, p_i)\}_{i=1}^{M} \subseteq \mathcal{D}_{\text{V}}$ denote the verified experience bank. Using normalized question representations, we retrieve the m nearest verified references $\mathcal{R}(q) = \text{TopM}_{(q_i,p_i)\in\mathcal{B}} \cos(h_q(q), h_q(q_i)), m \leq 8$, after removing exact question matches. Retrieval depends only on the test question and does not access its gold formulation.

Update-free curvature-aware reference alignment. At inference time, we use the retrieved verified experiences to select among multiple executable candidates without updating model parameters. To assess whether selected synthetic experiences provide useful signals beyond generation confidence, we study logit-, embedding-, first-order gradient-, and second-order curvature-based selection. Here, we present the curvature-aware method (FREE), with the remaining variants deferred to the appendix Sec A.3.

For a query q, let $\mathcal{R}(q) = \{(q_i, p_i)\}_{i=1}^{m}$ denote its retrieved verified references. For each reference, define $g_i = \nabla_\theta \ell(f_\theta(q_i), p_i)|_{\theta=\theta_2}$, and collect $G_{\mathcal{R}} = [g_1^\top; \ldots; g_m^\top]$ and $\bar{g}_{\mathcal{R}} = \frac{1}{m}\sum_{i=1}^{m} g_i$. Motivated by the curvature-preconditioned gradient interaction underlying RRInf (Tu et al., 2025), we construct the regularized empirical gradient second-moment matrix $H_{\mathcal{R}} = \frac{1}{m} G_{\mathcal{R}}^\top G_{\mathcal{R}} + \lambda I$, which serves as a local curvature surrogate over the retrieved references. For each executable candidate $c \in \mathcal{E}(q)$, let $g(q, c) = \nabla_\theta \ell(f_\theta(q), c)|_{\theta=\theta_2}$ and define $s_{\text{FREE}}(c \mid q) = \bar{g}_{\mathcal{R}}^\top H_{\mathcal{R}}^{-1} g(q, c)$. A larger score indicates stronger curvature-preconditioned alignment between the candidate gradient and the aggregate gradient direction of the retrieved verified references. We then select $c^\star = \arg\max_{c\in\mathcal{E}(q)} s_{\text{FREE}}(c \mid q)$. The inverse is not formed explicitly; the corresponding regularized linear system is solved using the retrieved reference gradients. Unlike standard influence functions, which estimate how upweighting a training example changes a reference objective, s_{FREE} does not estimate a parameter update. Instead, it measures the local curvature-aware compatibility between a generated candidate and retrieved verified OR experiences while keeping θ_2 fixed. RELIVE-OR reuses verified experience for both *training-time re-selection* and *update-free test-time program selection* via FREE.

3 BASELINES

Existing OR modeling systems mainly differ in how they construct training data and improve reasoning. **OptiMUS** focuses on inference-time formulation, execution, and repair (AhmadiTeshnizi et al., 2024). **ORLM**, **ReSocratic**, **OptMATH**, and **AutoOR** expand training data through scenario augmentation, formulation-based generation, back-translation, or predefined optimization templates (Huang et al., 2025b; Yang et al., 2025; Lu et al., 2025; Motwani et al., 2026). RELIVE-OR instead diversifies executable formulations while constructing verified natural-language–program pairs that preserve the originating OR model specification. Other methods focus on post-construction learning: **Step-Opt** progressively increases problem difficulty (Wu et al., 2025), whereas **SIRL** and **DeepOR** use solver feedback and reinforcement learning to improve formulation correctness and reasoning (Chen et al., 2026; Xiao et al., 2026). Additional related work is provided in Appendix F.

4 EXPERIMENTAL RESULTS

Table 1 reports results on six established autoformulation benchmarks—NL4Opt, NL4LP, MAMO-Easy, MAMO-Complex, IndustryOR, and ComplexOR (Ramamonjison et al., 2023; 2022; Huang et al., 2025c;b; Xiao et al., 2024). RELIVE-OR$_{\text{SFT}}$ achieves a reported macro-average of **71.4%**, outperforming Chain-of-Experts (**64.6%**), ORLM (**65.5%**), OR-R1$_{\text{SFT}}$ (**61.6%**), OR-R1$_{\text{RL}}$ (**65.1%**), and AutoOR$_{\text{SFT}}$ (**70.3%**). Following the reporting convention of AutoOR$_{\text{SFT}}$, whose available IndustryOR result is Oracle@3, this average uses the corresponding IndustryOR Oracle@3 result; the strict First-only average is **69.4%**. RELIVE-OR also exhibits substantial test-time scal-

Table 1: **Accuracy on six established benchmarks.** For RELIVE-OR, we report *First / SF@3 / Oracle@3*, denoting single-generation accuracy, self-consistent aggregation over three generations, and Oracle@3, respectively.

Method	Model	NL4OPT	NL4LP	MAMO-Easy	MAMO-Cplx	IndustryOR	ComplexOR	Avg.†
Prompting-based / Training-free Methods								
Standard	GPT-4o	61.2	73.6	70.3	57.7	38.1	42.9	57.3
CoT	GPT-4o	62.2	74.7	49.5	42.3	40.5	39.2	51.4
Chain-of-Experts	GPT-4o	66.7	87.4	94.4	50.6	31.2	57.1	64.6
CAFA	GPT-4o	68.1	50.0	71.2	44.5	41.1	46.4	53.6
Learning-based OR Models								
ORLM	8B	73.8	76.4	90.4	59.5	42.9	50.0	65.5
Step-Opt	8B	84.5	–	85.3	61.6	36.4	–	–
OptMATH	32B	95.9	–	89.9	54.1	–	–	–
OR-R1$_{\mathrm{SFT}}$	8B	86.0	82.9	87.0	39.9	33.0	40.7	61.6
OR-R1$_{\mathrm{RL}}$	8B	88.3	84.6	86.1	49.9	35.3	46.3	65.1
DeepOR$_{\mathrm{RL}}$	8B	97.7	82.9	93.2	67.1	52.2	64.3	76.2
AutoOR$_{\mathrm{SFT}}$	8B	66.7	82.9	84.0	74.0	64.3[‡]	50.0	70.3
AutoOR$_{\mathrm{RL}}$	8B	78.6	97.2	94.0	83.0	69.1	72.2	82.4
Larger General-Purpose Models								
GPT-OSS	120B	88.1	97.1	98.0	60.0	69.1	61.1	78.9
DeepSeek V4 Pro	–	81.0	88.6	92.0	34.0	73.8	72.2	73.6
Llama 4 Maverick	–	78.6	97.1	86.0	28.0	61.9	61.1	68.8
Llama 3.3	70B	61.9	77.1	60.0	8.0	31.0	44.4	47.1
Kimi K2.5	–	78.6	100.0	90.0	34.0	76.2	77.8	76.1
Ours-SFT								
RELIVE-OR$_{\mathrm{SFT}}$	**8B**	**81.7 / 86.9 / 88.3**	**91.0 / 96.6 / 97.2**	**90.6 / 94.9 / 96.9**	**54.1 / 73.9 / 78.4**	**54.8 / 61.9 / 66.7**[‡]	**44.4 / 55.6 / 61.1**	**71.4 / 78.3 / 81.4**
Ours+Update-Free								
RELIVE-OR$_{\mathrm{SFT}}$+FREE-Mean LogLik	**8B**	**87.32**	**96.07**	**95.60**	**65.77**	**54.76**	**61.11**	**76.8**
RELIVE-OR$_{\mathrm{SFT}}$+FREE-Trusted Emb.	**8B**	**87.79**	**97.19**	**95.41**	**70.27**	**52.38**	**61.11**	**77.4**
RELIVE-OR$_{\mathrm{SFT}}$+FREE-Gradient Align.	**8B**	**87.79**	**95.51**	**95.41**	**73.87**	**57.14**	**55.6**	**77.6**
RELIVE-OR$_{\mathrm{SFT}}$+FREE-Curvature Align.	**8B**	**86.85**	**96.07**	**94.50**	**69.37**	**54.76**	**55.6**	**76.19**

ing potential. SF@3 and Oracle@3 achieve macro-averages of **78.3%** and **81.4%**, respectively, compared with the strict First-only average of **69.4%**. The gains are particularly large on MAMO-Complex (**54.1** $\rightarrow$ **73.9** $\rightarrow$ **78.4%**) and ComplexOR (**44.4** $\rightarrow$ **55.6** $\rightarrow$ **61.1%**). Moreover, SF@3 (**78.3%**) exceeds DeepOR$_{\mathrm{RL}}$ (**76.2%**) without reinforcement learning and approaches GPT-OSS-120B (**78.9%**). These results indicate that correct formulations are often already present among generated candidates, making effective selection a key remaining bottleneck.

Efficient update-free specification-aware selection. FREE exploits the substantial First-to-Oracle gap without additional parameter updates. Using the same trained model and generated candidates, FREE-Mean LogLik, FREE-Trusted Embedding, and FREE-Gradient Alignment achieve **76.8%**, **77.4%**, and **77.6%**, respectively. FREE-Gradient Alignment improves the strict First-only average from **69.4%** to **77.6%**, recovering approximately **68%** of the gap to the Oracle@3 upper bound (**81.4%**). It also exceeds DeepOR$_{\mathrm{RL}}$ (**76.2%**) and approaches GPT-OSS-120B (**78.9%**) with only an 8B model. Importantly, FREE requires no iterative rollout–reward–policy-update loop: verified OR experiences are reused directly as inference-time selection signals. The improvement from likelihood to reference- and gradient-based selection further indicates that trusted OR model specifications provide useful information beyond generation confidence, supporting their role not only in synthetic-data construction and re-selection, but also in efficient test-time program selection. Experimental details and seed construction are provided in Appendices C and C.1.

5 Conclusion

RELIVE-OR treats synthetic OR data as *modeling experiences* whose value can change as the learner evolves. Rather than merely scaling the number of training examples, RELIVE-OR seeks to *maximize structural freedom while preserving OR model specification*. Its DE-SCALE component expands experience directly in executable formulation space, while closed-loop verification controls semantic, code-structural, and outcome consistency. Because verified experiences may still be redundant, imbalanced, or misaligned with the capabilities represented by the trusted seeds, RELIVE-OR further revisits them after broad consolidation. Retrospective influence measures their effect on a trusted seed objective and prioritizes experiences that remain useful to the current learner, thereby promoting OR model-specification consistency beyond generation-time verification. The same verified experience bank is subsequently reused at test time as a non-parametric reference memory for update-free candidate selection. Overall, RELIVE-OR separates three questions that are often conflated in synthetic OR modeling: Can we generate structurally diverse experiences? Are they reliable? Are they still useful to the current learner? This experience-centric view provides a unified framework for synthesizing, validating, revisiting, and reusing OR modeling experience.

AI USE STATEMENT

We used generative AI tools for language editing, improving clarity and presentation, and assisting with code development and debugging. All AI-assisted text, code, and experimental outputs were manually reviewed and verified by the authors. Research claims, methodological decisions, experimental design, and interpretation of results were determined and validated by the authors. We take full responsibility for the final content of this work, including any text, code, or artifacts produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the RELIVE-OR pipeline, training and inference procedures, datasets, evaluation protocols, and implementation settings in the main paper and appendix. We report the configurations required to reproduce our synthesis, verification, retrospective re-selection, and update-free candidate-selection experiments. Code, generated data, and experimental artifacts will be released upon publication to facilitate reproduction of the reported results.

REFERENCES

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

I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*, pp. 368–377, 1998. doi: 10.1109/ICSM.1998.738528.

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.

Frederick Hillier and Gerald Lieberman. *Introduction To Operations Research*, volume 139. 01 1969. doi: 10.2307/2345190.

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

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, 2025b.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. Llms for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 2678–2710, 2025c.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pp. 1885–1894. Pmlr, 2017.

David A. Kolb. *Experiential Learning: Experience as the Source of Learning and Development*. Prentice-Hall, Englewood Cliffs, NJ, 1984. ISBN 0132952610.

Sirui Li, Janardhan Kulkarni, Ishai Menache, Cathy Wu, and Beibin Li. Towards foundation models for mixed integer linear programming. In *International Conference on Learning Representations*, volume 2025, pp. 88590–88638, 2025.

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

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

Sumeet Ramesh Motwani, Chuan Du, Aleksandar Petrov, Christopher E. Davis, Philip Torr, Antonio R. Papania-Davis, and Weishi Yan. AutoOR: Scalably post-training LLMs to autoformulate operations research problems. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=BRISZRO6Qh.

Atsushi Nitanda, Anzelle Lee, Damian Tan, Mizuki Sakaguchi, and Taiji Suzuki. Propagation of chaos for mean-field langevin dynamics and its application to model ensemble. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org, 2025.

Yuangang Pan, Ivor W. Tsang, Weijie Chen, Gang Niu, and Masashi Sugiyama. Fast and robust rank aggregation against model misspecification. *Journal of Machine Learning Research*, 23(23): 1–35, 2022. URL http://jmlr.org/papers/v23/20-315.html.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. In *Advances in Neural Information Processing Systems*, volume 33, pp. 19920–19930, 2020.

Rindra Ramamonjison, Haley Li, Timothy Yu, Shiqi He, Vishnu Rengan, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. Augmenting operations research with auto-formulation of optimization models from problem descriptions. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 29–62, 2022.

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

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*, 2020.

Yewei Song, Cedric Lothritz, Daniel Tang, Tegawendé Bissyandé, and Jacques Klein. Revisiting code similarity evaluation with abstract syntax tree edit distance. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 38–46, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-short.3. URL https://aclanthology.org/2024.acl-short.3/.

Zhuozhuo Tu, Cheng Chen, and Yuxuan Du. RRInf: Efficient influence function estimation via ridge regression for large language models and text-to-image diffusion models. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 18494–18507, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.933. URL https://aclanthology.org/2025.emnlp-main.933/.

Yang Wu, Yifan Zhang, Yurong Wu, Yuran Wang, Junkai Zhang, and Jian Cheng. Training LLMs for optimization modeling via iterative data synthesis and structured validation. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 12880–12896, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.691. URL https://aclanthology.org/2025.findings-emnlp.691/.

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

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. doi: 10.1609/aaai.v40i40.40699.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. OptiBench Meets ReSocratic: Measure and Improve LLMs for Optimization Modeling. In *The Thirteenth International Conference on Learning Representations*, 2025.

Jack Yurkiewicz and Pace Plaza. Operations research: Applications and algorithms, by wayne l. winston, duxbury press, boston, 1987, 1025 pages. *Networks*, 18(3):254–256, 1988. doi: https://doi.org/10.1002/net.3230180310. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230180310.

A Appendix

A.1 Notation

Tables 2 and 3 summarize the main notation used throughout the paper. We use ϕ for the synthesis and reconstruction model and θ for the OR model being trained.

A.2 Overview of RELIVE-OR

The Figure 3 shows the details of Data-Efficient Scenario-Conditioned StructurAl Latent SynthEsis (DE-SCALE).

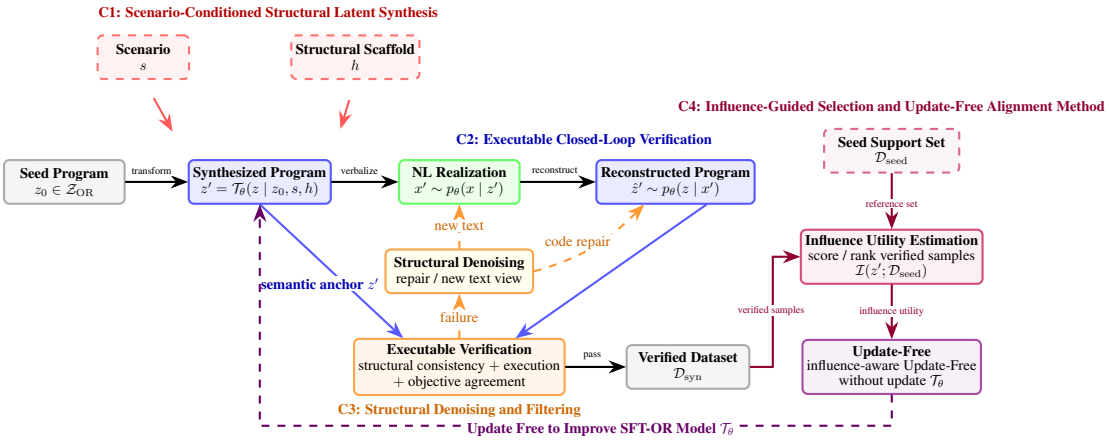

Figure 3: Overview of our framework. **C1:** A scenario s and structural scaffold h guide the transformation of a seed program z_0 into a synthesized program z'. **C2:** z' is verbalized into x' and reconstructed as $\hat{z}'$, then verified for structural consistency, executability, and objective agreement. **C3:** Failed samples undergo bounded code repair or alternative text realization; verified pairs form $\mathcal{D}_{\mathrm{syn}}$ for broad consolidation. **C4:** After SFT, verified experiences are retrospectively ranked by their influence on the trusted seed set and selectively revisited for focused consolidation; at inference time, FREE further reuses verified experiences for update-free program selection.

A.3 Details of Reference-Guideed Update-Free Test Time Program Selection (FREE)

Response-loss gradients. To ensure a common scoring protocol, each candidate program c is placed in a canonical <CODE>...</CODE> response conditioned on q. Likewise, every retrieved verified program p_i is conditioned on its own question q_i.

Let

$$\ell_\theta(q, p) = -\frac{1}{|p|} \sum_{t=1}^{|p|} \log P_\theta\left(p_t \mid q, p_{<t}\right) \tag{19}$$

Table 2: **Notation for data, synthesis, structure, and validation.**

Notation	Definition
OR experiences and distributions	
x	Natural-language description of an OR problem.
z	Solver-executable formulation code corresponding to x.
$r = (x, z)$	An OR modeling experience represented as a problem–code pair.
$\mathcal{D}_0 = \{r_i^0 = (x_i^0, z_i^0)\}_{i=1}^{N_0}$	Validated trusted seed set.
N_0	Number of trusted seed experiences.
$P^\star$	Unknown target distribution of correctly specified OR modeling experiences.
$\mathcal{D}_{\text{syn}} = \{r_j = (x'_j, z'_j)\}_{j=1}^{N}$	Pool of synthesized OR experiences before verification.
N	Number of synthesized candidate experiences.
Q_{syn}	Distribution over experiences induced by the synthesis process.
$\mathcal{D}_V$	Verified synthetic experience pool.
Q_V	Distribution induced by the verified experience pool $\mathcal{D}_V$.
OR structure and specification	
$\psi(r)$	Structural attributes of an OR experience, such as variable types, objective structure, constraint families, and their relations.
$\mathcal{C}_0$	Structural coverage represented by the trusted seed set.
$\mathcal{C}^\star = \text{supp}(\psi_\# P^\star)$	Structural support of the target OR distribution.
$\text{Spec}(r_i^0)$	Seed-derived OR specification associated with trusted experience r_i^0.
S_i^0	Shorthand for $S_i^0 = \text{Spec}(r_i^0)$.
$\mathcal{F}_{\text{spec}}(S_i^0)$	Set of admissible OR experiences preserving specification S_i^0.
$\pi(j)$	Index of the trusted seed from which synthesized experience r_j originates.
Structural synthesis and validation	
s	Scenario specifying a new decision setting for synthesis.
h	Structural scaffold providing generic OR modeling guidance.
ϕ	Parameters of the LLM used for synthesis, verbalization, and reconstruction.
T_ϕ	Scenario-conditioned structural code synthesis operator.
$z' \sim T_\phi(z \mid z^0, s, h)$	Synthesized executable formulation code derived from seed code z^0.
$p_\phi(x \mid z)$	Code-to-text conditional generation distribution.
$p_\phi(z \mid x)$	Text-to-code reconstruction distribution.
$\hat{z}'$	Executable formulation reconstructed from synthesized problem x'.
$\mathcal{S}(z', \hat{z}')$	Indicator of structural consistency between synthesized and reconstructed code.
$\mathcal{E}(z)$	Indicator that code z executes successfully and returns a finite objective.
$f(z)$	Objective value returned by executing formulation code z.
$V(z', \hat{z}')$	Single-view verification criterion.
K_{view}	Maximum number of independently generated textual views used for verification.
$V_{K_{\text{view}}}(r_j)$	Multi-view verification indicator for synthesized experience r_j.

denote mean response-token negative log-likelihood, where prompt tokens are excluded from the loss.

We compute gradients with respect to a fixed parameter subset $\theta_S \subseteq \theta$:

$$\boxed{g(q,p) = \nabla_{\theta_2} \ell_\theta(q,p).} \tag{20}$$

In our implementation, S matches `norm.weight`, typically corresponding to the final normalization weight. Importantly, these gradients are computed only as response representations:

$$\boxed{\Delta\theta = 0.} \tag{21}$$

No optimizer step is performed.

Gradient-guided ranking. Each candidate program c is placed in a canonical `<CODE>...</CODE>` response conditioned on q. Each retrieved program p_i is evaluated

Table 3: **Notation for consolidation, retrospective influence, and update-free selection.**

Notation	Definition
Broad and focused consolidation	
$\mathcal{D}_{\text{train}} = \mathcal{D}_0 \cup \mathcal{D}_V$	Training set used for broad consolidation.
$\mathcal{D}_R \subseteq \mathcal{D}_V$	Retrospectively selected subset used for focused consolidation.
B_{retro}	Retrospective selection budget, i.e., $\lvert\mathcal{D}_R\rvert$.
f_θ	OR model mapping a natural-language problem to formulation code.
θ	Parameters of the OR model being trained.
$\ell(f_\theta(x), z)$	Training loss for problem–code pair (x, z).
θ_1	Model parameters after broad consolidation on $\mathcal{D}_{\text{train}}$.
θ_2	Model parameters after focused consolidation on $\mathcal{D}_R$.
Retrospective influence	
$L_0(\theta)$	Trusted reference objective evaluated on $\mathcal{D}_0$.
$\ell_j(\theta)$	Loss associated with verified synthetic experience r_j.
$g_0 = \nabla_\theta L_0(\theta_1)$	Gradient of the trusted reference objective at θ_1.
$g_j = \nabla_\theta \ell_j(\theta_1)$	Gradient induced by verified synthetic experience r_j.
$R_{\text{train}}(\theta)$	Average training risk over $\mathcal{D}_{\text{train}}$.
$H_1 = \nabla^2_\theta R_{\text{train}}(\theta_1)$	Hessian of the broad-training objective at θ_1.
$I_0(r_j)$	Influence of upweighting r_j on the trusted reference objective.
G_1	Empirical second moment of per-example gradients used by RRInf.
λ	Damping parameter in the RRInf approximation.
$\widetilde{I}_0(r_j)$	Damped RRInf influence target for experience r_j.
$\widehat{I}_0(r_j)$	Numerically estimated retrospective influence of r_j.
$S_{\text{retro}}(r_j) = -\widehat{I}_0(r_j)$	Retrospective utility score; larger values indicate greater predicted utility.
Update-free inference-time selection	
q	Unseen natural-language OR problem at inference time.
$\mathcal{C}(q)$	Set of formulation-code candidates generated for query q.
K_{gen}	Number of generated candidates for each query.
$\mathcal{E}(q)$	Subset of generated candidates that execute successfully.
$\mathcal{B} \subseteq \mathcal{D}_V$	Verified experience bank used as non-parametric reference memory.
$R(q)$	Retrieved verified references for query q.
m	Number of retrieved reference experiences.
$h_q(\cdot)$	Question encoder used for reference retrieval.
g_i	Gradient associated with retrieved reference experience i.
G_R	Matrix collecting gradients of retrieved reference experiences.
$\bar{g}_R$	Mean gradient over retrieved reference experiences.
H_R	Regularized local curvature matrix estimated from retrieved-reference gradients.
$g(q, c)$	Gradient associated with executable candidate c for query q.
$s_{\text{FREE}}(c \mid q)$	Curvature-aware reference-alignment score assigned to candidate c.
$c^\star$	Candidate selected by maximizing $s_{\text{FREE}}(c \mid q)$.

in the same format conditioned on its own question q_i. Let

$$L_\theta(q, p) = -\frac{1}{|p|} \sum_{t=1}^{|p|} \log P_\theta(p_t \mid q, p_{<t}) \tag{22}$$

denote the mean negative log-likelihood over response tokens, excluding prompt tokens. We represent each problem–program pair by its response-loss gradient over a fixed parameter subset θ_S:

$$\boxed{g(q, p) = \nabla_{\theta_2} L_\theta(q, p).} \tag{23}$$

In our implementation, S matches `norm.weight`, typically the final normalization weight.

For candidate c, the gradient-alignment score is

$$\boxed{s_{\text{grad}}(c \mid q) = \frac{1}{m} \sum_{(q_i, p_i) \in \mathcal{R}(q)} \frac{g(q, c)^\top g(q_i, p_i)}{\|g(q, c)\|_2 \, \|g(q_i, p_i)\|_2 + \epsilon}.} \tag{24}$$

Stage	Input → Output	Operation
Experience synthesis	$\mathcal{D}_0 \to \mathcal{D}_{\text{syn}}$	Perturb executable OR formulations and reconstruct corresponding problem–program experiences to increase structural coverage.
Integrated validation	$\mathcal{D}_{\text{syn}} \to \mathcal{D}_{\text{V}}$	Retain experiences satisfying semantic, code-structural, outcome, and seed-relative model-specification consistency.
Broad consolidation	$\mathcal{D}_0 \cup \mathcal{D}_{\text{V}} \to \theta_1$	Fine-tune on the complete verified experience set to acquire broad OR modeling capability.
Retrospective valuation	$(\theta_1, \mathcal{D}_0, \mathcal{D}_{\text{V}}) \to \widehat{I}_0$	Use the trusted seeds as a reference objective and estimate, via RRInf, the marginal utility of additional exposure to each verified synthetic experience.
Focused consolidation	$\widehat{I}_0 \to \mathcal{D}_{\text{R}} \to \theta_2$	Select experiences with the greatest predicted benefit to the current learner and continue SFT from θ_1.
Update-free selection	$(q, \theta_2, \mathcal{D}_{\text{V}}) \to c^\star$	Generate multiple executable programs, retrieve related verified OR experiences, and rank candidates by response-gradient alignment without updating model parameters.

The selected program is

$$c^\star_{\text{grad}} = \arg\max_{c \in \mathcal{E}(q)} s_{\text{grad}}(c \mid q), \tag{25}$$

with ties broken by generation order.

Controlled selectors. We compare the gradient-based methods against two update-free baselines while holding the model, generated candidates, eligible set, and retrieved reference set fixed.

Mean log-likelihood uses

$$s_{\text{LL}}(c \mid q) = -\ell_\theta(q, c). \tag{26}$$

For the trusted-embedding baseline, let $h_p(q, p)$ denote the normalized response/program representation. We define

$$s_{\text{Emb}}(c \mid q) = \frac{1}{m} \sum_{(q_i, p_i) \in \mathcal{R}(q)} \cos\left(h_p(q, c), h_p(q_i, p_i)\right). \tag{27}$$

For every selector s, the prediction is

$$c^\star_s = \arg\max_{c \in \mathcal{E}(q)} s(c \mid q), \tag{28}$$

with ties resolved by generation order.

B END-TO-END RELIVE-OR PIPELINE

End-to-end RELIVE-OR pipeline. Specification verification controls pair-level synthesis errors, while retrospective learning addresses learner-dependent utility after broad consolidation. Verified experiences are additionally reused as test-time references without parameter updates.

C EXPERIMENTAL DETAILS

For synthetic data generation, we use DeepSeek-V3 Liu et al. (2024) with a temperature of 0.7 and top-p sampling with $p = 0.8$. All generated optimization programs are formulated using IBM DOcplex and executed and verified using the IBM CPLEX optimizer. We have used A100 80G GPUS for our experiments.

Table 4: Technical progression of synthetic-data construction for optimization modeling. p, m, and c denote the natural-language problem, mathematical formulation, and executable solver program, respectively.

Method	**Technical progression / verification**
ORLM (Huang et al., 2025b)	GPT-4 rephrases questions, modifies objectives and constraints together with the corresponding formulation/code, or introduces alternative modeling techniques. Generated programs are mainly corrected and filtered by executability, without an independent $p' \to \hat{c}$ reconstruction check.
ReSocratic (Yang et al., 2025)	Diversity is introduced jointly in natural-language reasoning and mathematical formulations, after which the synthesized demonstration is translated into a question and executable code; the executable program itself is not the primary object of structural evolution.
MILP-Evolve	Each MILP class is represented as modular executable code consisting of *Data* D, *Optimization Modeling* M, and *Parameters* Θ. The evolved classes are filtered through syntax checks, parameter search, feasibility, and solver-property criteria, but not through language-based reconstruction.
OptMATH (Lu et al., 2025)	It starts from curated mathematical formulations and mainly varies problem data and instance complexity within seed-defined formulation families. Back-translation and forward modeling provide rejection-based verification, but the seed executable structure is not first evolved by an LLM into a new formulation structure.
RELIVE-OR	**RELIVE-OR combines LLM-driven executable structural evolution with multi-view code–text–code reconstruction.** Reconstructed programs are checked for structural, execution, and numerical consistency, while learning consistency further controls whether the transformed experience preserves its intended learning role.

Table 5: Technical progression of synthetic-data construction and learning for optimization modeling.

Method	**Synthesis, verification, and learning pipeline**
ORLM (Huang et al., 2025b)	$(p, m, c) \xrightarrow[\text{scenario expansion / rephrasing / objective–constraint / technique augmentation}]{\text{LLM}} (p', m', c') \xrightarrow{\text{execution filtering}} \mathcal{D}_{\text{syn}} \xrightarrow{\text{SFT}} f_\theta$
ReSocratic (Yang et al., 2025)	$(p, m) \xrightarrow[\text{demonstration synthesis}]{\text{LLM}} (p', m') \xrightarrow[\text{question + code generation}]{\text{LLM}} (q', c') \xrightarrow{\text{SFT}} f_\theta$
MILP-Evolve	$c = (G, \mathcal{M}, \Theta) \xrightarrow[\text{Add / Delete / Mutate Crossover / New}]{\text{LLM structural evolution}} c' = (G', \mathcal{M}', \Theta') \xrightarrow[\text{parameter search}]{\text{syntax + solver filtering}} I' \xrightarrow[\text{integrality gap / strong-branching action}]{\text{solver supervision}} y' \xrightarrow{\text{GNN training}} f_\theta^{\text{GNN}}$
OptMATH (Lu et al., 2025)	$m \xrightarrow[\text{instance generation}]{\text{parameterization}} d' \xrightarrow[\text{back-translation}]{\text{LLM}} p' \xrightarrow[\text{forward modeling}]{\text{LLM}} \hat{c} \xrightarrow{\text{rejection verification}} \mathcal{D}_{\text{syn}} \xrightarrow{\text{SFT}} f_\theta$
DeepOR (Xiao et al., 2026)	$p \xrightarrow[\text{flowchart-guided CoT synthesis}]{\text{LLM}} (\omega, c) \xrightarrow{\text{SFT}} f_\theta^{\text{reason}} \xrightarrow[\text{solver feedback + modeling checklist}]{\text{GRPO + hard-case SFT}} f_\theta^{\text{DeepOR}}$
AutoOR (**?**)	$T \xrightarrow[\text{standard-form instantiation}]{\text{LLM}} w = (f, \{g_i\}, x^*, c^*, \mu) \xrightarrow[\text{back-translation}]{\text{LLM}} p \xrightarrow[\text{component-wise check against } w]{\text{LLM verification}} (p, c^*) \xrightarrow[R_{\text{exec}} + R_{\text{feas}} + R_{\text{opt}}]{\text{GRPO}} f_\theta : p \to c$
RELIVE-OR	$c_0 \xrightarrow[\text{scenario-guided structural evolution}]{\text{LLM}} c' \xrightarrow[\text{multi-view verbalization}]{\text{LLM}} \{p'_k\}_{k=1}^K \xrightarrow[\text{reconstruction + repair}]{\text{LLM}} \{\hat{c}'_k\}_{k=1}^K \xrightarrow[\text{structure + execution + numerical consistency}]{\text{closed-loop verification}} \mathcal{D}_{\text{verified}} \xrightarrow{\text{learning consistency}} \mathcal{D}_{\text{train}} \xrightarrow{\text{LLM training}} f_\theta : p \to c$

Notation.

- p denotes a natural-language optimization problem; m a mathematical formulation; c executable solver code; and (p', m', c') their synthesized or transformed counterparts. q' denotes a synthesized natural-language question, and $\hat{c}$ denotes solver code independently reconstructed from generated text. $\mathcal{D}_{\text{syn}}$ is a synthetic training dataset and f_θ is the resulting trained LLM.
- For MILP-Evolve, $c = (G, \mathcal{M}, \Theta)$ denotes a modular executable MILP class, where G is the data-generation module, $\mathcal{M}$ is the optimization-modeling module containing variables, objective, and constraints, and Θ contains instance-generation parameters. $c' = (G', \mathcal{M}', \Theta')$ is the LLM-evolved class, I' is an instantiated MILP generated from the evolved class, y' denotes solver-derived supervision such as an integrality-gap target or strong-branching action, and f_θ^{GNN} is the trained GNN.
- For OptMATH, d' denotes generated numerical problem data or parameters. For DeepOR, $\omega = (z_0, \ldots, z_n)$ denotes a synthesized multi-step reasoning trajectory, f_θ^{reason} the reasoning model after expertise SFT, and f_θ^{DeepOR} the model after subsequent self-improvement learning.
- For AutoOR, T denotes a predefined optimization standard-form template and w its instantiated *world descriptor*; f is the concrete objective, $\{g_i\}$ the instantiated constraints, x^* the ground-truth optimal solution, c^* the ground-truth executable solver code, and μ auxiliary metadata describing variables, units, domains, and structural information. R_{exec}, R_{feas}, and R_{opt} denote execution, feasibility, and optimality rewards used for GRPO.
- For RELIVE-OR, c_0 denotes the mentor/seed executable program, c' its structurally evolved program, $\{p'_k\}_{k=1}^K$ are K independently generated natural-language views of c', and $\{\hat{c}'_k\}_{k=1}^K$ are the corresponding reconstructed and repaired executable programs. $\mathcal{D}_{\text{verified}}$ denotes experiences passing structural, execution, and numerical consistency checks, while $\mathcal{D}_{\text{train}}$ denotes the subset retained after learning consistency assessment. Finally, $f_\theta : p \to c$ denotes an LLM trained to map a natural-language optimization problem to executable solver code.

Table 6: Comparison of synthetic dataset scale for optimization modeling. Bold values highlight the size of the synthesized and filtered datasets.

Method	Initial Seeds	After Synthesis	After Filtering	Used for Training
ORLM Huang et al. (2025a)	686	**32,481**	32,481	32,481
OptMATH Lu et al. (2025)	53 generators	**>600,000**	Rejection sampling	∼200,000+
Method	**Seeds Sample**	**After Synthesis**	**After Verification**	**Used for Training**
DE-SCALE (Round 1)	**48 of 199**	**23,900**	**Repair and Reconstruction**	**14,272**
DE-SCALE (Round 2)	**119**	**8,377**	**Repair and Reconstruction**	**5,060**
DE-SCALE (Round 3)	**119**	**10681**	**Repair and Reconstruction**	**8,102**
DE-SCALE (Influence)	**1287**	**-**	**-**	**-**
Total	**1406 Huang et al. (2025a)**	**-**	**-**	**-**

C.1 Seed Problems and Data Construction

We use a total of **119 manually curated OR seed problems** as the starting point for synthetic data generation. Among these, 54 are derived from examples in OptMATH Lu et al. (2025), while the remaining seeds are collected from optimization textbooks and online educational resources.

For the 54 OptMATH-derived seeds, we retain only the natural-language problem descriptions and do not use their original generators or executable implementations. Given each natural-language OR problem, we independently use Claude and ChatGPT in chat mode to construct solver-executable formulations. The resulting natural-language/code pairs are manually validated and used as the initial seeds for our scenario-guided synthesis pipeline.

Each validated seed is further associated with a *structural scaffold* that specifies generic optimization-modeling principles without exposing problem-specific variables, parameter values, objective functions, or constraints. Consequently, when generating a new scenario, the model must infer the corresponding optimization formulation from the synthesized problem description rather than directly copying the formulation of the seed.

Summary of Modeling Techniques in the Collected Operations Research Questions

We manually collected **48** operations research (OR) questions from *Operations Research* by Wayne L. Winston (4th edition) Yurkiewicz & Plaza (1988) and *Introduction to Operations Research* by Frederick S. Hillier and Gerald J. Lieberman (9th edition) Hillier & Lieberman (1969). These questions contain a range of modeling constructs that require non-trivial formulation choices. We assign each question to one mutually exclusive category according to the modeling techniques required in its reference formulation.

Among the 48 questions:

- **16** involve *Big–M constraints with indicator variables*;
- **15** involve *indicator variables only*;
- **9** involve *piecewise-linear functions*;
- **2** involve both *indicator variables and piecewise-linear functions*;
- **1** involves a *small-ϵ refinement*; and
- **5** require none of the above modeling constructs.

Executable competence and candidate support. To examine whether the compact synthetic corpus provides effective supervision beyond aggregate benchmark accuracy, we report end-to-end accuracy, executable coverage, and Oracle@8 across six benchmarks. DE-SCALE achieves an average Overall Accuracy of 55.4% with 90.6% Coverage, indicating that the model can produce executable solutions for most instances while maintaining competitive end-to-end performance. Excluding NL4OPT, Overall Accuracy remains 48.9% with 89.3% Coverage, showing that the result is not driven solely by the relatively easier benchmark. Oracle@8 further reaches 62.2% on average and 56.4% without NL4OPT, demonstrating that correct executable solutions are present in the candidate set more often than they are ultimately selected. Together, these results suggest that our compact, structurally diverse synthetic corpus provides effective supervision for recovering executable

Modeling technique	**Count**	**% of 48**
Big–M constraints with indicator variables	16	33.3%
Indicator variables only	15	31.3%
Piecewise-linear functions	9	18.8%
Indicator variables with piecewise-linear functions	2	4.2%
Small-ϵ refinement	1	2.1%
None of the above	5	10.4%
Total	**48**	**100.0%**

Table 7: Distribution of modeling techniques across the 48 manually collected operations research questions. Categories are mutually exclusive.

Modeling technique	**Question IDs (Q55–Q102)**	**Count**
Big–M constraints with indicator variables	Q55, Q57, Q61, Q65, Q66, Q67, Q69, Q70, Q76, Q77, Q78, Q80, Q89, Q97, Q98, Q100	16
Indicator variables only	Q56, Q62, Q68, Q71, Q72, Q73, Q74, Q75, Q79, Q81, Q82, Q91, Q93, Q95, Q99	15
Piecewise-linear functions	Q59, Q63, Q83, Q84, Q85, Q86, Q92, Q96, Q101	9
Indicator variables with piecewise-linear functions	Q58, Q94	2
Small-ϵ refinement	Q88	1
None of the above	Q60, Q64, Q87, Q90, Q102	5
Total		**48**

Table 8: Question-level categorization of the modeling techniques in the collected OR seed set.

decision structures, while further improvements remain possible through more reliable candidate selection and execution.

C.2 Evaluation Metrics

We have provided the evaluation metrics table in 18. Definition of evaluation metrics. Let N denote the number of benchmark instances, $K = 3$ the number of generated candidates, $y_i^\star$ the gold objective value, $y_i^{(k)}$ the executed objective value of candidate k, and y_i^{sel} the prediction selected by executable consensus. A prediction is valid if it executes successfully and returns a finite numerical answer.

D Examples of Synthetic Samples

Structural diversity. We evaluate structural diversity on 6,539 randomly sampled synthesized training examples by pairing each variant with its originating seed. Rather than relying on textual similarity, we compare executable Python ASTs and normalized structural ASTs Song et al. (2024), where identifiers and literal values are removed while optimization-relevant program structure is preserved. All sampled variants differ from their seeds at the executable-AST level. Moreover, 95.47% are classified as structural formulation variants and a further 4.53% exhibit a detected shift in broad OR formulation family. The mean structural-AST similarity is 0.585, suggesting that the generated examples retain some relationship to their seeds while substantially altering their optimization structure. No sampled examples are classified as identical code, surface-only rewrites, or simple instance-level perturbations.

Example of structural synthesis. Table 13 illustrates a representative seed-to-variant transformation. The original seed is a long–short portfolio optimization problem whose decision variables are signed asset weights and whose objective trades expected return against quadratic portfolio risk. Its constraints encode dollar neutrality, gross leverage, turnover, individual weight limits, and tracking error. In contrast, the synthesized problem introduces non-negative inventory-reserve variables and an auxiliary capacity variable, changes the objective from return maximization with quadratic risk

Table 9: Composition of the seed corpus ($n = 119$) by application family and model class. MIP covers all seeds with at least one integer or binary variable (pure IP/BIP and mixed MILP); QP/SOCP covers the quadratic and second-order-cone portfolio seeds; NLP is one non-convex seed solved by enumeration.

Application family	LP	MIP	QP/SOCP	NLP	Total
Production planning & lot sizing	15	16	0	1	32
Transportation & network flow	9	3	0	0	12
Facility location & network design	0	11	0	0	11
Assignment & matching	1	7	0	0	8
Set covering & partitioning	0	8	0	0	8
Machine & project scheduling	0	8	0	0	8
Portfolio & conic finance	0	0	8	0	8
Knapsack & capital budgeting	0	7	0	0	7
Blending & diet	5	0	0	0	5
Cutting stock, packing & partitioning	1	3	0	0	4
Routing (TSP / VRP / transit)	0	4	0	0	4
Resource allocation (other)	0	4	0	0	4
Energy dispatch & unit commitment	1	2	0	0	3
Revenue management & market share	1	2	0	0	3
Facility dispersion	0	2	0	0	2
Total	33	77	8	1	119

Table 10: Provenance blocks in the seed corpus. "Variant metadata" means the seed carries non-empty `scenario_forcing`, `new_variables_vs_base` and `new_constraints_vs_base` fields, i.e. it is directly usable by the variant generator.

Block	IDs	n	Seed style	Variant meta.	Emb. demo.
A	1–54	54	Seeded `Generator` class, cited source	✓	✓
B	55–102	48	Fixed-data script, short technique hint	–	✓
C	103–109	7	Raw CPLEX API, Cholesky lifting, 2–4 cones	✓	✓
D	110–119	10	`seed_data` dict + scaffold key	–	–
	All	119		61	109

penalization to linear cost minimization, removes the portfolio-specific constraints, and introduces a minimum-reserve requirement together with a second-order cone capacity constraint. Therefore, the generated example changes the underlying executable decision structure rather than merely modifying coefficients or paraphrasing the original problem.

Qualitative evidence of structural diversity. Table 14 illustrates that the proposed synthesis procedure can generate substantially different optimization structures from a validated seed. The original problem is a two-period portfolio rebalancing formulation involving 20 primary portfolio weights, inter-period turnover coupling, covariance-dependent tracking-error constraints, and portfolio-risk epigraphs. The synthesized problem instead introduces three inventory-reserve decisions, removes the temporal and portfolio-specific structure, replaces the return–turnover–risk objective with a linear holding-cost objective, and introduces a minimum-reserve constraint together with a norm-capacity constraint. Importantly, both problems remain continuous conic formulations. Thus, the transformation changes the *decision structure and formulation family* without merely switching solver classes or perturbing numerical parameters.

Qualitative evidence of formulation-level synthesis. Table 15 provides a representative example of the structural diversity induced by our synthesis procedure. The original seed is a constrained portfolio-allocation problem involving ten asset weights, turnover variables, benchmark-relative factor-exposure constraints, a covariance-based tracking-error constraint, and a variance epigraph. The synthesized variant instead introduces three inventory-reserve decisions, removes all portfolio- and factor-specific relationships, replaces the return–turnover–risk objective with a linear reserve-cost objective, and introduces minimum-reserve, item-specific bound, and second-order cone con-

Table 11: Structural-diversity analysis of randomly sampled synthesized training examples. Each synthesized variant is compared against its corresponding seed. AST-based metrics remove superficial formatting effects, while formulation-level metrics examine changes in optimization variables, objectives, constraints, modeling constructs, and OR problem families.

Metric	Measurement	Result	Rate
Assessable seed–variant pairs	Randomly sampled synthesized examples paired with their corresponding original seed.	6,539	100%
New executable AST	Exact comparison of parsed Python abstract syntax trees (ASTs); formatting and comments do not affect the executable AST.	6,539 / 6,539	**100.00%**
Structural formulation variant	Comparison after normalizing identifiers and literal values. A variant is structural if variables, objective structure, constraints, comparison structure, modeling constructs, or model class change.	6,243 / 6,539	**95.47%**
OR formulation-family shift	Rule-based detection of a change between broad OR families, together with changes in at least two major formulation components.	296 / 6,539	**4.53%**
Identical code	Raw executable code is identical to the corresponding seed.	0	0.00%
Surface rewrite only	Source text differs, but the parsed executable AST remains identical.	0	0.00%
Instance-only variant	Executable AST changes, but the normalized formulation skeleton remains unchanged; differences are primarily values, dimensions, or identifiers.	0	0.00%

Table 12: Representative structural formulation-family shifts. Lower structural similarity indicates larger structural deviation from the corresponding seed.

Seed	Variant	Sim.	Family Shift
104	356 Table [8]	0.287	Portfolio → Production
106	5 Table [9]	0.312	Portfolio → Production
103	356 Table [10]	0.315	Portfolio → Production

straints. The two executable programs have a structural similarity of only 0.315. This demonstrates that the synthesized example is not a numerical perturbation or linguistic paraphrase of the seed, but a transformation of the underlying optimization formulation.

E Multi-Text Generations

Multi-text cycle consistency substantially reduces representation-induced false rejection. Using five alternative textual reconstructions recovers 13 additional known-good question-code pairs, increasing verification success from 30.25% to 41.18% (+10.92 pp; +36.1% relative) and reducing the false-rejection rate from 69.75% to 58.82%. Notably, this improvement is achieved without mismatch-code repair, indicating that diversification of the intermediate semantic representation itself accounts for the recovery.

E.1 Model Fine-tuning

We fine-tune the base model using a LoRA ensemble consisting of $M = 7$ independently optimized adapters, following the parameter-merging strategy of Nitanda et al. (2025). For a pretrained weight matrix $\mathbf{W}_0 \in \mathbb{R}^{k\times d}$, each LoRA adapter introduces a rank-N update

$$\Delta\mathbf{W}_j = \mathbf{B}_j\mathbf{A}_j, \tag{29}$$

where $\mathbf{A}_j \in \mathbb{R}^{N\times d}$ and $\mathbf{B}_j \in \mathbb{R}^{k\times N}$. The adapted weight is therefore

$$\mathbf{W}_j = \mathbf{W}_0 + \gamma\mathbf{B}_j\mathbf{A}_j, \qquad \gamma > 0, \tag{30}$$

where $\mathbf{W}_0$ remains frozen and only the low-rank parameters $\mathbf{A}_j$ and $\mathbf{B}_j$ are optimized.

We independently optimize the M LoRA parameter sets $\{(\mathbf{A}_j, \mathbf{B}_j)\}_{j=1}^{M}$ using noisy AdamW. At optimization step k, Gaussian perturbation $\sqrt{2\lambda\eta_k}\,\boldsymbol{\xi}_k$ is added to the parameter update, where η_k denotes the step size, $\boldsymbol{\xi}_k$ is standard Gaussian noise, and λ controls the noise magnitude. After fine-tuning, the M adapters are merged into a single low-rank update. Writing the rank-one components of adapter j as $\mathbf{b}_j^i(\mathbf{a}_j^i)^\top$, the merged update is

$$\Delta\mathbf{W} = \frac{1}{MN}\sum_{j=1}^{M}\sum_{i=1}^{N}\mathbf{b}_j^i(\mathbf{a}_j^i)^\top. \tag{31}$$

We use $N = 32$ LoRA rank, $M = 7$ adapters, $\lambda = 10^{-5}$, and fine-tune each adapter for three epochs. The merged update is subsequently applied to the frozen pretrained weights for downstream evaluation.

Thus, we can apply PoC-based model ensemble for LoRA parameters. Note that the ensemble model is reduced to a single $(k \times d)$-matrix ΔW due to the linearity of the activation function, and therefore it does not require additional memory and time for inference.

F Related Work

LLMs for optimization modeling and data synthesis. LLMs have increasingly been used to translate natural-language decision problems into mathematical formulations and executable solver programs. **OptiMUS** focuses on inference-time formulation, code generation, execution, and repair (AhmadiTeshnizi et al., 2024). More recent work constructs synthetic data for training specialized OR models. **ORLM** generates problem–formulation–code triples through scenario expansion and targeted augmentation (Huang et al., 2025b), while **ReSocratic** modifies natural-language descriptions and mathematical formulations before converting them into executable programs (Yang et al., 2025). **OptMATH** starts from curated mathematical formulations and generates instances of varying scale and complexity, using back-translation and forward modeling for verification (Lu et al., 2025). **AutoOR** instantiates predefined optimization templates, generates corresponding natural-language descriptions, and trains models with solver-based rewards (Motwani et al., 2026). RELIVE-OR differs by treating the executable formulation itself as the primary object of diversification. Its DE-SCALE component transforms variables, domains, objectives, constraints, and their relationships before generating the corresponding natural-language problem. The transformed program then serves as an anchor in a code–text–code reconstruction loop. Closest in spirit, **MILP-Evolve** also evolves executable MILP programs through operations such as mutation, addition, deletion, and crossover (Li et al., 2025). However, its evolved instances are used primarily for training GNN-based MILP solvers, whereas RELIVE-OR constructs verified natural-language–program pairs for LLM autoformulation and explicitly checks whether the synthesized problem preserves the OR model specification of its originating seed.

Training and reasoning for optimization modeling. Other work focuses on improving training or reasoning once OR examples have been constructed. **Step-Opt** progressively generates harder optimization problems with structured validation (Wu et al., 2025). **SIRL** uses solver feedback as a reinforcement-learning reward to encourage executable and correct programs (Chen et al., 2026), while **DeepOR** combines structured reasoning traces, SFT, and GRPO to improve optimization-modeling reasoning (Xiao et al., 2026). These approaches primarily improve problem difficulty, output correctness, or reasoning behavior. RELIVE-OR instead focuses on the life cycle of the modeling experience itself. Structural synthesis first expands the executable formulation space; integrated validation removes experiences that are semantically, code-structurally, or numerically invalid, as well as those that violate the OR model specification encoded by their originating seeds. The resulting experiences are then used for broad consolidation and subsequently revisited through *retrospective learning*, which asks which already-valid experiences remain useful to the current learner. Finally, the verified experience bank is reused at inference time for update-free, reference-guided program selection. Thus, RELIVE-OR separates three questions: *Can we create structurally diverse experiences? Are they valid realizations of the intended OR specification?* and *Are they still useful after the learner has adapted?*

Table 13: Example of a scenario-guided structural transformation. Starting from a validated portfolio-optimization seed, our synthesis procedure generates an executable contingency-inventory problem with different decision variables, objective, constraints, and application semantics. This example illustrates that the generated sample is not merely a numerical perturbation or linguistic paraphrase of the seed.

	Original Seed: Portfolio Optimization	**Synthesized Variant: Contingency Inventory**
Scenario	Long–short portfolio allocation across $N = 10$ stocks under risk, leverage, turnover, and tracking-error requirements.	Reserve planning for three inventory items with additional capacity to absorb possible reallocation demand.
Decision variables	Signed continuous portfolio weights ω_i, together with auxiliary variables for leverage, turnover, tracking error, and variance.	Non-negative continuous reserve quantities r_1, r_2, r_3 and additional capacity h.
Objective	$\max_{\omega} \; f^\top \omega - \lambda \omega^\top \Sigma \omega$ Maximize expected return while penalizing portfolio risk.	$\min_{r_1,r_2,r_3,h} \; 2r_1 + 3r_2 + 2.5r_3 + h$ Minimize inventory-reserve and additional-capacity cost.
Core constraints	$\sum_i \omega_i = 0,$ $\sum_i \lvert\omega_i\rvert \le L,$ $-0.5 \le \omega_i \le 0.5,$ $\sum_i \lvert\omega_i - \omega_{0,i}\rvert \le \delta,$ $(\omega - \omega_b)^\top \Sigma (\omega - \omega_b) \le \sigma^2.$	$r_1 + r_2 + r_3 \ge 25,$ $r_1, r_2, r_3, h \ge 0,$ $(0.5r_1)^2 + (0.4r_2)^2 + (0.6r_3)^2 \le h^2.$
Optimization structure	Portfolio QCP with quadratic risk objective, absolute-value constraints, and tracking-error constraint.	SOCP with a linear cost objective, minimum-reserve constraint, and second-order cone constraint.
Structural change	**Variables + objective sense + objective structure + constraint structure + application/formulation family**	

Table 14: Example of scenario-guided structural synthesis. The synthesized problem preserves the broad continuous conic optimization class while substantially changing the application domain, decision variables, objective, temporal structure, and constraint system.

	Original Seed: Two-Period Portfolio Rebalancing	**Synthesized Variant: Inventory Reserve Planning**
Scenario	Allocate a portfolio over two periods while controlling turnover, tracking error, and portfolio risk.	Determine non-negative reserves for three inventory items subject to a minimum total reserve and a norm-capacity requirement.
Primary variables	Portfolio weights $\omega_{1,i}, \omega_{2,i} \in [0, 0.3]$ for $N = 10$ assets.	Continuous reserve quantities $r_1, r_2, r_3 \ge 0$.
Auxiliary variables	Turnover variables $u_{1,i}, u_{2,i}$, Cholesky lifting variables $z_{1,j}, z_{2,j}$, risk epigraphs y_1, y_2, and cone radii t_1, t_2.	Scaled reserve variables y_1, y_2, y_3 and fixed norm-budget variable $t = 8$.
Objective	$\min \; -f_1^\top \omega_1 - f_2^\top \omega_2 + \sum_i u_{1,i} + \sum_i u_{2,i} + y_1 + y_2$	$\min \; 2r_1 + 3r_2 + 2.5r_3$
Core constraints	$\sum_i \omega_{1,i} = 1, \quad \sum_i \omega_{2,i} = 1,$ $\sum_i u_{1,i} \le 0.10, \quad \sum_i u_{2,i} \le 0.10,$ $u_{1,i} \ge \lvert\omega_{1,i} - \omega_{0,i}\rvert,$ $u_{2,i} \ge \lvert\omega_{2,i} - \omega_{1,i}\rvert,$ $\lVert L^\top (\omega_p - \omega_b)\rVert_2 \le \sqrt{0.025},$ $\omega_p^\top \Sigma \omega_p \le y_p^2, \qquad p \in \{1, 2\}.$	$r_1 + r_2 + r_3 \ge 25,$ $y_1 = 0.5r_1,$ $y_2 = 0.4r_2,$ $y_3 = 0.6r_3,$ $t = 8,$ $y_1^2 + y_2^2 + y_3^2 \le t^2.$
Temporal structure	Two-stage structure with explicit coupling between ω_1 and ω_2 through period-2 turnover.	Single-stage optimization with no temporal coupling.
Problem data	Two expected-return vectors, a covariance matrix, initial portfolio, benchmark portfolio, and turnover/risk budgets.	Three holding costs, three scaling coefficients, a total-reserve requirement, and a fixed norm budget.
Formulation family	Portfolio finance / multi-period portfolio rebalancing.	Inventory reserve / capacity planning.
Broad model class	Continuous conic/QCP formulation.	Continuous conic/SOCP formulation.

Table 15: Representative example of scenario-guided structural synthesis. The synthesized variant changes the application domain, decision variables, objective structure, and constraint system of the original executable seed, while remaining within continuous conic optimization.

	Original Seed: Constrained Portfolio Optimization	**Synthesized Variant: Inventory Reserve Planning**
Scenario	Allocate weights across 10 stocks to maximize risk-adjusted return while controlling turnover, benchmark deviation, factor exposures, portfolio variance, and tracking error.	Determine reserve quantities for three inventory items while satisfying minimum reserve, item-specific bounds, and a conic capacity requirement.
Primary variables	Portfolio weights ω_i, $i = 1, \ldots, 10$, with $0.05 \le \omega_i \le 0.15$.	Reserve quantities $r_1, r_2, r_3 \ge 0$, with $r_1 \le 10$ and $r_3 \ge 4$.
Auxiliary variables	Turnover variables u_i, Cholesky variables z_j, variance epigraph y, and fixed tracking-error radius t.	Cone variables z_1, z_2, z_3 and fixed cone-radius variable $t = 8$.
Objective	$\min \; -\sum_{i=1}^{10} f_i \omega_i + \sum_{i=1}^{10} C_i u_i + y$	$\min \; 2r_1 + 3r_2 + 2.5r_3$
Core constraints	$\sum_i \omega_i = 1,$ $0.05 \le \omega_i \le 0.15,$ $\sum_i u_i \le 0.1,$ $u_i \ge \lvert\omega_i - \omega_{0,i}\rvert,$ $\underline{m}_k \le M_k^\top \omega \le \overline{m}_k,$ $\underline{q}_k \le I_k^\top \omega \le \overline{q}_k,$ $\lVert L^\top (\omega - \omega_{\text{bench}})\rVert_2 \le 0.1414,$ $\omega^\top \Sigma \omega \le y.$	$r_1 + r_2 + r_3 \ge 25,$ $r_1 \le 10,$ $r_3 \ge 4,$ $z_1 = 0.5r_1,$ $z_2 = 0.4r_2,$ $z_3 = 0.6r_3,$ $z_1^2 + z_2^2 + z_3^2 \le 8^2.$
Domain-specific structure	Benchmark-relative portfolio allocation with market and industry factor-exposure constraints.	Inventory-capacity planning with minimum stock and item-specific reserve bounds.
Formulation family	Portfolio finance / constrained asset allocation.	Inventory reserve / capacity planning.
Structural similarity	0.315 (OR_FORMULATION_FAMILY_SHIFT)	

Table 16: Structural-diversity analysis on randomly sampled synthesized training examples. Each synthesized variant v_i is compared with its corresponding seed s_i. $\mathcal{A}(\cdot)$ denotes the executable AST, $\widetilde{\mathcal{A}}(\cdot)$ the normalized structural AST, $F_k(\cdot)$ one of eight formulation components, and $f(\cdot)$ the detected OR formulation family. AST-based comparison follows established structural code-analysis practice Baxter et al. (1998); Ren et al. (2020), while the OR-specific formulation metrics are introduced in this work.

Metric	Equation	What It Measures	Reference	Result
Executable AST Novelty	$I^{(i)}_{\text{exec}} = \mathbb{I}[\mathcal{A}(s_i) \neq \mathcal{A}(v_i)]$, $R_{\text{exec}} = \frac{1}{N}\sum_{i=1}^{N} I^{(i)}_{\text{exec}}$	Whether the parsed executable structure differs from the seed; formatting and comments alone do not count.	AST-based comparison Baxter et al. (1998); Ren et al. (2020)	**100.0%**
Structural-AST Similarity	$S_{\text{struct}}(s_i, v_i) = \frac{2M(\widetilde{\mathcal{A}}(s_i), \widetilde{\mathcal{A}}(v_i))}{\lvert\widetilde{\mathcal{A}}(s_i)\rvert + \lvert\widetilde{\mathcal{A}}(v_i)\rvert}$	Similarity after normalizing identifiers and literal values while preserving operators, API calls, control flow, and executable structure.	Baxter et al. (1998); Ren et al. (2020)	**Mean: 0.585**
Structural Formulation Novelty	$I^{(i)}_{\text{struct}} = \mathbb{I}\Big[C_i \geq 1 \vee \Big(\widetilde{\mathcal{A}}(s_i) \neq \widetilde{\mathcal{A}}(v_i) \wedge S_{\text{struct}}(s_i, v_i) < 0.995\Big)\Big]$	Identifies changes in optimization formulation structure beyond simple numerical, dimensional, or identifier perturbations.	**Ours**; AST foundation Baxter et al. (1998); Ren et al. (2020)	**100.0%**
Same-Family Structural Variant	$I^{(i)}_{\text{SF}} = \mathbb{I}\Big[I^{(i)}_{\text{struct}} = 1 \wedge f(s_i) = f(v_i)\Big]$	Structural modification of the optimization formulation while remaining within the same detected broad OR family.	**Ours**	**95.47%**
OR Formulation-Family Shift	$I^{(i)}_{\text{family}} = \mathbb{I}[f(s_i) \neq f(v_i) \wedge C_i \geq 2]$	Conservative cross-family diversity requiring both a detected OR-family change and at least two changed formulation components.	**Ours**	**4.53%**

Table 17: Comparison of given code and direct projection to question-to-code generation, legacy one-text cycle verification, and multi-text cycle verification on 119 known-good question–code pairs.

Metric	Legacy / Baseline	Multi-Text
Known-good pairs	119	119
Source executable	119 (100.00%)	119 (100.00%)
Original question-to-code pass	47 (39.50%)	47 (39.50%)
Cycle-consistency pass	36 (30.25%)	**49 (41.18%)**
False rejection rate	69.75%	**58.82%**
Recovered by multi-text	–	**13**
Absolute success-rate gain	–	**+10.92 pp**
Relative success-rate gain	–	**+36.1%**
Text candidates per sample	1	5
Reverse codes per text	1	1
Minimum reverse passes per text	1	1
Mismatch-code repairs per text	0	0

Table 18: Definition of evaluation metrics. Let N denote the number of benchmark instances, $K = 8$ the number of generated candidates, $y_i^\star$ the gold objective value, $y_i^{(k)}$ the executed objective value of candidate k, and y_i^{sel} the prediction selected by executable consensus. A prediction is valid if it executes successfully and returns a finite numerical answer.

Metric	Definition	Interpretation
Code Rate	$\frac{N_{\mathrm{code}}}{N}$	Fraction of benchmark instances for which executable code can be extracted from at least one generated output.
Execution Success	$\frac{N_{\mathrm{exec}}}{N}$	Fraction of instances for which the generated code executes successfully and produces a usable numerical prediction.
Coverage	$\frac{N_{\mathrm{covered}}}{N}$	Fraction of benchmark instances for which the final inference procedure returns a valid finite numerical prediction.
Covered Accuracy	$\frac{N_{\mathrm{correct}}}{N_{\mathrm{covered}}}$	Accuracy conditional on successfully returning a valid prediction. Invalid, failed, missing, or non-finite predictions are excluded from the denominator.
Overall / Selected@3 Accuracy	$\frac{1}{N}\sum_{i=1}^{N}\mathbb{I}\left[y_i^{\mathrm{sel}} \simeq y_i^\star\right]$	End-to-end system accuracy after executable consensus selection among $K = 3$ candidates. Invalid, failed, or uncovered predictions count as incorrect.
Oracle@3	$\frac{1}{N}\sum_{i=1}^{N}\mathbb{I}\left[\exists k \in \{1,\ldots,3\} : y_i^{(k)} \simeq y_i^\star\right]$	Fraction of problems for which at least one of the three generated candidates is correct. Measures recoverable candidate-level potential, not deployable accuracy.