

MILP-Solve-Free Selection of LLM-Generated MILP Programs via Multi-Signal Structural-Semantic Consistency

Shuli Zeng¹, Guozhu Liu¹, Yinghao Chen¹, Wantong Xie¹, Ji Liu¹,
Shaojie Tang², Feng Wu^{1*}, Sijia Zhang¹, Xiang-Yang Li^{1*}

¹University of Science and Technology of China, Hefei, China

²University at Buffalo, Buffalo, NY, USA

{zengshuli0130, liuguozhu, chenyinghao, wantongxie, liuji4474}@mail.ustc.edu.cn

shaojiet@buffalo.edu; wufeng02@ustc.edu.cn

sxzs@ustc.edu.cn; xiangyangli@ustc.edu.cn

Abstract

Large language models make natural-language-to-MILP modeling increasingly plausible, but reliably selecting among multiple generated programs remains challenging: structurally similar errors can create false majorities, while solving every candidate integer program is costly. We propose Structural-Semantic Consistency (SSC), a best-of- K selector combining structural consistency, static quality, text alignment, and two diagnostics obtained from continuous LP relaxations. Notably, SSC does not require solving any candidate integer programs during selection. Across seven benchmarks and 618 fixed candidate pools, SSC obtains 55.7% accuracy, compared with a random baseline of 51.5%, 53.7% for a zero-shot listwise LLM judge, and 55.3% for a leave-one-benchmark-out learned ranker. The symbolic fusion reaches 53.6%, with LP diagnostics adding 2.1 percentage points. These results establish multi-signal weak verification as a competitive and interpretable baseline for candidate selection.

1 Introduction

Optimization problems are ubiquitous in real-world decision-making, from logistics and production planning to scheduling, energy management, and resource allocation. Mixed-Integer Linear Programs (MILPs) provide a powerful and widely used modeling language for such problems (Nemhauser and Wolsey, 1988), but turning an informal business or scientific description into an executable formulation still requires both domain knowledge and solver-facing modeling expertise (Li et al., 2023).

Large language models (LLMs) can reduce this modeling burden by translating natural-language problem descriptions into executable Pyomo implementations of LP/MILP models (Ahmaditeshnizi et al., 2024; Ramamonjison et al., 2022; Xiao et al.,

2024). However, individual generations remain unreliable (Yang et al., 2025; Lu et al., 2025; Chen et al., 2026). Thus, reliability depends not only on generating candidate formulations but also on selecting among them: given multiple candidate formulations for the same problem, which one should be trusted?

A direct execution-based strategy is to solve every candidate MILP and compare its optimal objective value against a reference answer. However, this approach is unsuitable for deployment-time selection: solving all K candidate integer programs can be computationally expensive (Snell et al., 2024); gold reference answers are unavailable at deployment; and syntactically valid Pyomo programs may encode infeasible, unbounded, or ill-posed models that return no useful solver signal. These barriers motivate our central question: can we reliably select an answer-correct LLM-generated MILP program without solving each candidate integer program?

Existing solver-feedback methods operate in a different regime. OptiMUS (Ahmaditeshnizi et al., 2024) solves candidate MILPs during generation to provide feedback for iterative refinement, while OptiTree (Liu et al., 2025) invokes a solver at each tree-search node to evaluate partial formulations. Such methods require solver access throughout generation and can incur repeated solver calls across candidates and refinement rounds. In contrast, SSC operates on a fixed candidate pool and performs post-generation selection without solving any candidate integer programs.

Figure 1 illustrates both the potential and the challenges of fixed-pool selection. On OptMath (Lu et al., 2025), first-candidate accuracy is 41%, while pool coverage reaches 75%, indicating substantial room for improvement if the selector can reliably identify an answer-correct candidate. However, simple agreement is unreliable: candidates often share libraries, APIs, and surface code

*Corresponding authors.

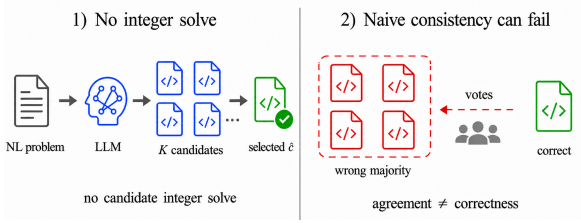


Figure 1: Motivation and problem setting. Best-of- K generation creates substantial headroom for same-pool selection, as random expected accuracy remains below the pool-oracle upper bound. However, answer-incorrect candidates with similar structures can form false-majority clusters, rendering naive consistency-based selection unreliable.

patterns, and near-duplicate answer-incorrect programs can form false-majority clusters that appear more consistent than the correct minority.

We propose **Structural-Semantic Consistency (SSC)**, a no-integer-solve best-of- K selector for fixed candidate pools. SSC combines three symbolic signals—structural consistency (SC), static quality (SQ), and text alignment (TA)—with two LP-relaxation diagnostics based on relaxed-solution and relaxed-objective agreement. The LP diagnostics provide lightweight executable evidence, but SSC neither solves candidate integer programs nor serves as a correctness verifier. Across seven NL-to-MILP benchmarks, SSC achieves 55.7% answer accuracy, compared with a random expectation of 51.5%, 53.7% for a zero-shot listwise LLM judge, and 55.3% for a leave-one-benchmark-out learned ranker. The LP-relaxation diagnostics improve accuracy by 2.1 percentage points (pp) over the three symbolic signals alone.

Our contributions are threefold:

- **False-majority failures in MILP-solve-free NL-to-MILP selection.** We show that naive self-consistency is insufficient for best-of- K selection: near-duplicate, answer-incorrect candidates can form false-majority clusters, causing structural consistency to favor incorrect solutions.
- **SSC: a symbolic-relaxation selector for brittle candidate pools.** We introduce SSC, which integrates three symbolic signals—structural consistency, static quality, and text alignment—with bounded LP-relaxation diagnostics. This design mitigates false-majority failures in self-consistency while detecting defects that structural signals alone may miss.

- **MILP-solve-free selection across seven benchmarks.** Across seven NL2MILP benchmarks, SSC reaches 55.7% answer accuracy on a fixed no-reference-answer pool. It is competitive with zero-shot LLM judging and leave-one-benchmark-out learned fusion; the symbolic selector alone achieves 53.6%, with LP-relaxation diagnostics contributing an additional 2.1 pp.

2 Problem Formulation

Let $\mathcal{D}$ be a benchmark of natural-language optimization problems, and let $\mathcal{P}^p \in \mathcal{D}$ denote the problem indexed by p . For each $\mathcal{P}^p$, an LLM generates a pool of K candidate programs $\mathcal{C}^p = \{c_1^p, \dots, c_K^p\}$. Each c_i^p is a Pyomo (Hart et al., 2011; Bynum et al., 2021) implementation of a MILP model through a `build_model()` interface.

For offline evaluation, let v_i^p be the finite objective value returned by the frozen solver protocol for candidate c_i^p , and let $v_\star^p$ be the gold answer. A candidate that returns no valid objective is incorrect; otherwise, we label c_i^p correct if

$$|v_i^p - v_\star^p| \leq \max(\epsilon_{\text{rel}} \max(|v_i^p|, |v_\star^p|), \epsilon_{\text{abs}}), \quad (1)$$

where $\epsilon_{\text{rel}} = \epsilon_{\text{abs}} = 10^{-4}$. This correctness label is used only for offline evaluation and is never provided to the selector.

A selector is a function

$$s : (\mathcal{P}^p, \mathcal{C}^p) \rightarrow \mathcal{C}^p, \quad (2)$$

which chooses one candidate from the pool. In this work, selection is performed under a no-integer-solve constraint: during selection, the selector may inspect source code, parsed model structure, problem text, and lightweight diagnostics, but it must not solve candidate MILPs or invoke integer optimization. This setting isolates selection from generation and studies whether a candidate can be identified using only non-integer evidence under a limited verification budget.

For benchmark $\mathcal{D}$, selector accuracy measures the fraction of problems for which the selected candidate is correct:

$$\text{Acc}(\mathcal{D}, s) = \frac{1}{|\mathcal{D}|} \sum_{p \in \mathcal{D}} \mathbf{1}[\text{correct}(\hat{c}^p)], \quad (3)$$

where $\hat{c}^p = s(\mathcal{P}^p, \mathcal{C}^p)$ denotes the candidate selected by s for problem p . We also report pool coverage, which measures whether the generated

pool contains at least one correct candidate:

$$\text{Coverage}(\mathcal{D}, K) = \frac{1}{|\mathcal{D}|} \sum_{p \in \mathcal{D}} \mathbf{1}[\exists i \in [K] \text{ s.t. } \text{correct}(c_i^p)]. \quad (4)$$

Coverage upper-bounds the accuracy achievable by any selector operating on the same candidate pool. Because a selector cannot succeed when all generated candidates are incorrect, $\text{Acc}(\mathcal{D}, s) \leq \text{Coverage}(\mathcal{D}, K)$. This inequality distinguishes two failure modes. A generation failure occurs when the candidate pool contains no answer-correct program. A selection failure occurs when at least one answer-correct program exists, but the selector chooses an incorrect candidate. Formally,

$$\begin{aligned} \mathcal{E}_{\text{gen}} &= \{p \in \mathcal{D} : \forall i \in [K], \neg \text{correct}(c_i^p)\}, \\ \mathcal{E}_{\text{sel}}(s) &= \{p \in \mathcal{D} : \exists i \in [K], \text{correct}(c_i^p) \\ &\quad \wedge \neg \text{correct}(\hat{c}^p)\}. \end{aligned} \quad (5)$$

The total error therefore decomposes into generation and selection errors:

$$1 - \text{Acc}(\mathcal{D}, s) = \frac{|\mathcal{E}_{\text{gen}}|}{|\mathcal{D}|} + \frac{|\mathcal{E}_{\text{sel}}(s)|}{|\mathcal{D}|}. \quad (6)$$

For static comparison, each candidate c_i^p is parsed into a model signature

$$\mathcal{M}_i^p = (\mathcal{V}_i^p, \mathcal{O}_i^p, \mathcal{R}_i^p). \quad (7)$$

Here $\mathcal{V}_i^p$ is the set of variable signatures. Each variable signature records the variable name, domain type (e.g., binary, integer, or continuous), index arity, bound expressions, and tokens appearing in its definition. The objective signature $\mathcal{O}_i^p$ records the canonicalized objective sense and linear expression structure, including variable-shape tokens and numeric constants. The constraint-signature set $\mathcal{R}_i^p$ contains one signature for each standardized constraint row; each signature records canonical sense, linear terms, variable shapes, numeric magnitudes, and a coarse expression profile. For successfully constructed models, maximization objectives are mapped to canonical minimization and upper-bound rows are sign-flipped to canonical lower-bound form; original senses remain available to the text-alignment (TA) component. These signatures are obtained by constructing each Pyomo model in isolation and converting its expressions to Pyomo’s standard linear representation, with AST-based checks and implementation-specific fallback procedures. Constructing $\mathcal{M}_i^p$ neither invokes an optimizer nor solves the candidate MILP.

3 Method: Structural-Semantic Consistency

Figure 2 gives an overview of the full pipeline. Given a candidate pool $\mathcal{C}$ and a problem description $\mathcal{P}$, SSC combines three symbolic signals: structural consistency (SC), measuring agreement with other candidates; static quality (SQ), summarizing static checks; and text alignment (TA), measuring agreement with the problem text:

$$B_i^{\text{raw}} = 0.35 \text{SC}_i + 0.20 \text{SQ}_i + 0.45 \text{TA}_i. \quad (8)$$

For a candidate with a fatal construction or static defect, the symbolic base is capped by its static-quality score; otherwise the base score remains unchanged. The two LP diagnostics summarize agreement under continuous relaxation: LP-sol measures solution-profile consistency across candidates, while LP-obj measures proximity to the median feasible relaxed objective. Denoting the resulting symbolic score by B_i , the final SSC score is

$$\text{SSC}(c_i) = 0.65 B_i + 0.20 \widehat{\text{LP-sol}}_i + 0.15 \widehat{\text{LP-obj}}_i. \quad (9)$$

The hatted LP diagnostics are normalized within each candidate pool, and the candidate with the highest SSC score is selected. We describe the computation of each component in the following subsections. Exact edge-case and normalization conventions are given in Appendix A.

3.1 Structural Consistency

The structural consistency score of candidate c_i measures how similar its program structure is to the rest of the pool. Correct formulations tend to share variable types, constraint structure, and objective form; this structural agreement is a signal of correctness. However, consistency can be misleading when incorrect candidates form a dense cluster. Let $G_p = \{i : \text{correct}(c_i^p)\}$ and $B_p = \{i : \neg \text{correct}(c_i^p)\}$. A false majority occurs when

$$\begin{aligned} &\max_{i \in B_p} \sum_{\substack{j \in B_p \\ j \neq i}} \text{sim}(\mathcal{M}_i, \mathcal{M}_j) \\ &> \max_{i \in G_p} \sum_{\substack{j \in G_p \\ j \neq i}} \text{sim}(\mathcal{M}_i, \mathcal{M}_j). \end{aligned} \quad (10)$$

We define pairwise program similarity $\text{sim}(\mathcal{M}_i, \mathcal{M}_j) \in [0, 1]$ as a weighted combination of five sub-scores (variable, objective, constraint,

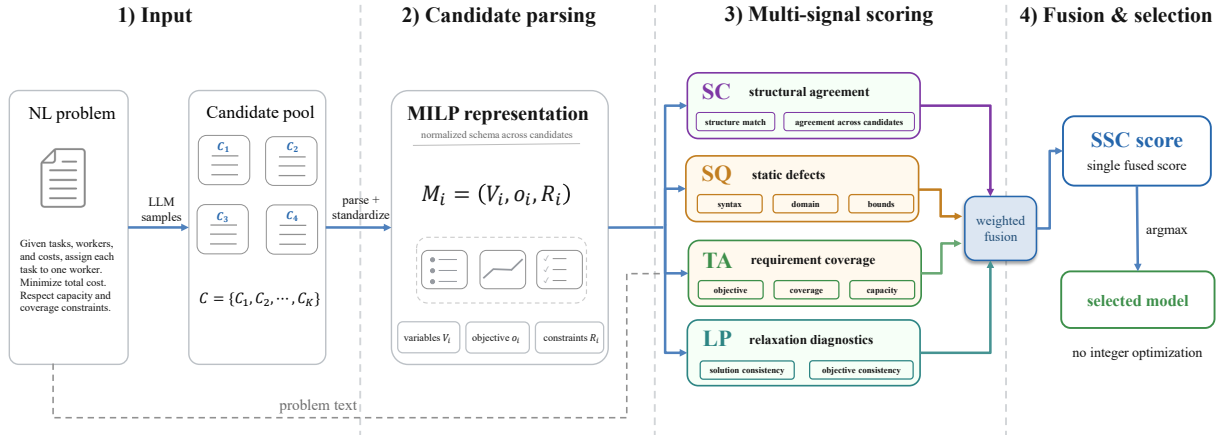


Figure 2: Overview of SSC: each candidate c_i is parsed into $\mathcal{M}_i = (\mathcal{V}_i, o_i, \mathcal{R}_i)$, scored with SC, SQ, TA, and LP-relaxation diagnostics, and ranked by the fused SSC score.

parameter-dependency, and shape similarity; see Appendix E for definitions and weights).

To define clone-aware weights, we construct an undirected graph $G_{\text{clone}} = ([K], E_{\text{clone}})$ over the K candidates, with an edge $(a, b) \in E_{\text{clone}}$ iff $\text{sim}(\mathcal{M}_a, \mathcal{M}_b) \geq \tau_{\text{clone}}$, where $\tau_{\text{clone}} = 0.95$. Let $[j]$ denote the set of candidate indices in the connected component of G_{clone} containing j (i.e., candidate c_j). The no-reference-answer experiments weight each candidate c_j by clone diversity, static quality, and text alignment,

$$w_j = \frac{\text{SQ}(c_j) \text{TA}(c_j, \mathcal{P})}{|[j]|}, \quad (11)$$

and define

$$\text{SC}(c_i) = \frac{\sum_{j \neq i} w_j \text{sim}(\mathcal{M}_i, \mathcal{M}_j)}{\sum_{j \neq i} w_j}. \quad (12)$$

This weighting reduces the influence of near-duplicate or low-credibility candidates, but SC remains imperfect: a cluster of structurally similar incorrect candidates can still dominate a correct minority.

3.2 Static Quality

Even without optimization, several quality defects are directly observable from the parsed representation: unknown variable domains, missing or ambiguous constraint senses, parse failures, or absent objectives. We define a penalty set $\mathcal{N}$ of named defect types, each with a weight $w_n \in [0, 1]$ reflecting its severity:

$$\text{SQ}(c_i) = \max \left(0, 1 - \sum_{n \in \mathcal{D}(\mathcal{M}_i)} w_n \right), \quad (13)$$

where $\mathcal{D}(\mathcal{M}_i)$ is the set of defects detected in c_i and $w_n \in [0, 1]$ is the severity weight of defect type n . In our experimental setup, SQ uses fixed severity weights shared across all benchmarks: fatal syntax errors are assigned weight 1.00, Pyomo API and construction failures are assigned weights from 0.35 to 0.70, missing variables or objectives are assigned 0.30, and softer semantic defects such as possible nonlinearity, unknown variable domains, and unknown constraint senses are assigned 0.15, 0.08, and 0.06, respectively.

3.3 Text Alignment

Text alignment measures the natural-language faithfulness of a candidate: i.e., the extent to which its symbolic content accurately reflects the problem description $\mathcal{P}$. This is the primary NLP component of SSC.

Let $\phi(\mathcal{M}_i)$ be the multiset of semantic tokens extracted from the parsed model (variable names, parameter identifiers, numeric magnitudes, constraint-sense labels, and linear-term tokens). Let $\psi(\mathcal{P}) = \{(f_k, u_k)\}$ be a set of weighted, closed-vocabulary semantic requirements extracted from $\mathcal{P}$ by an LLM call (Appendix L), where f_k is a required feature token and $u_k \in [0.1, 2.0]$ is its importance weight. The extraction is performed once per problem, shared by all K candidates, and cached across ablations; candidate scoring itself then uses only the closed-vocabulary requirements and rule-based evidence from parsed code. Text alignment is the weighted coverage of these requirements:

$$\text{TA}(c_i, \mathcal{P}) = \frac{\sum_{k: f_k \in \phi(\mathcal{M}_i)} u_k}{\sum_k u_k}. \quad (14)$$

TA signal quality. TA captures coarse require-

ments such as objective direction, variable domains, and constraint types. It does not verify constants, units, or arbitrary relational semantics; its role is therefore complementary rather than verificatory.

3.4 LP Solution Consistency

SSC solves the bounded continuous relaxation (Nemhauser and Wolsey, 1988; Wolsey, 1998) of each candidate. LP-sol constructs a permutation-invariant profile by sorting relaxed variable values and truncating or zero-padding to 30 dimensions, then averages non-negative cosine similarity over valid peers. LP-obj assigns higher scores to feasible objectives closer to the central feasible objective in the pool. Both diagnostics are normalized within the pool; failed or invalid relaxations contribute zero. GLPK (Free Software Foundation, 2023) uses a 5-second internal limit, with a process-level watchdog recorded in the run metadata. Because sorting removes variable identities and relaxation removes integrality, these signals are weak behavioral diagnostics rather than semantic alignment or correctness verification. Exact definitions are given in Appendix A; their contribution is analyzed in §5.2.

3.5 Candidate Generation

SSC is generation-agnostic. In our experiments, each problem is paired with a retrieved MILP template and $K=10$ candidates are generated using diverse formulation-emphasis prompts and a cycled temperature schedule. Candidates undergo up to two non-optimization repair rounds based on Pyomo construction checks; full generation details are provided in Appendix M.

4 Experiments

4.1 Benchmarks

We evaluate on seven benchmarks representing diverse optimization problem types and difficulty levels (Table 1).

These benchmarks span template-based linear programming descriptions, mathematical modeling problems, complex operations-research formulations, and industrial-style instances, covering a broad range of difficulty levels. Detailed per-benchmark descriptions—including source, task format, evaluation split, and difficulty rationale—are provided in Appendix B.

Benchmark	Total	Eval	Difficulty
NL4OPT (Ramamonjison et al., 2022)	230	100	Easy
MaMO-Easy (Huang et al., 2025b)	652	100	Easy
MaMO-Complex (Huang et al., 2025b)	211	100	Medium
ComplexOR (Xiao et al., 2024)	18	18	Hard
IndustryOR (Huang et al., 2025a)	100	100	Hard
OptiBench (Yang et al., 2025)	605	100	Hard
OptMath (Lu et al., 2025)	166	100	Hard

Table 1: Evaluation benchmarks. **Total**: full test-split size. **Eval**: problems used in our experiments; where for benchmarks with Total>100, we use the first 100 records from the normalized test file. All problems have numeric gold answers.

4.2 Baselines and Setup

We present baseline comparisons before ablations. First, we report published prompt-only Standard, CoT (Wei et al., 2022), and CoE (Xiao et al., 2024) results from OptiTree (Liu et al., 2025). They use DeepSeek-V3 (DeepSeek-AI, 2024) or GPT-4o under different generation protocols, so they provide performance context rather than a controlled comparison of selection methods. Second, our controlled baselines—random selection, a zero-shot listwise LLM judge, and a leave-one-benchmark-out (LOBO) learned ranker—select from the same $K=10$ pools as SSC. The judge receives the problem and candidate formulations but no solver output or reference answer; LOBO trains on candidate labels from six benchmarks and evaluates on the held-out benchmark.

Our candidates were generated on July 11, 2026 via the rolling API endpoint recorded in the manifests as deepseek-v4-flash (DeepSeek-AI, 2026), using five formulation-emphasis variants paired with temperatures [0.6, 0.8, 1.0, 1.2, 1.4] and repeated twice. SSC uses fixed weights shared across benchmarks and is evaluated offline with SCIP (Achterberg, 2009). Throughout, MILP-solve-free (equivalently, integer-solve-free) means that selection does not solve the candidate MILPs or run integer optimization. Full SSC does call an LP solver (GLPK (Free Software Foundation, 2023)) on continuous relaxations as a lightweight diagnostic; SCIP is invoked only for offline evaluation. We report **Random expected** (51.5%), the mean per-problem correct fraction across all K positions; and **Random draw (first position)**, the accuracy of selecting the deterministically shuffled first candidate (Appendix M.5). The two values can differ due to finite-sample variance; Random draw enables paired McNemar significance testing.

4.3 Metrics

We report answer accuracy under the offline correctness definition in §2. Candidates that fail to construct or return a valid objective value under the frozen evaluation protocol are marked incorrect. Correctness is determined by comparing the returned objective value with the reference answer. Pool coverage denotes the same-pool oracle upper bound: the fraction of problems whose K -candidate pool contains at least one answer-correct candidate.

4.4 Main Results

Published contextual baselines. Table 2 first situates SSC against previously published single-pass prompting results. These results provide useful field-level reference points, although differences in model versions, prompting strategies, sampling procedures, and candidate budgets make them not directly comparable to our fixed-pool experiment. We include our SSC row in the same table to provide a consolidated view of the numerical landscape, but we neither compute a joint ranking nor interpret these results as evidence of a controlled advantage for SSC.

Controlled same-pool baselines. Table 3 isolates the effect of selection by evaluating all methods on the same 6,180 candidates. SSC selects an answer-correct candidate for 344 of 618 problems (55.7%), compared with 51.5% random expectation, 53.7% for the zero-shot LLM judge, and 55.3% for LOBO learned fusion. SSC and LOBO differ on only ten problems (paired exact $p=0.754$); SSC versus the judge has 38 SSC-only and 26 judge-only successes (paired exact $p=0.169$). These results support SSC’s competitiveness with these complementary selectors, rather than a claim of general superiority. Relative to selecting the first candidate, SSC has 43 SSC-only and 21 first-only successes ($p=0.0087$); because design choices were informed by this pool, we treat this as descriptive paired evidence. Pool coverage is 66.0%, leaving 10.4 percentage points of same-pool selection headroom.

5 Analysis

5.1 Component Ablation

After the baseline comparisons, we examine the contribution of SSC’s individual internal signals through the ablation results in Figure 3 and Table 4.

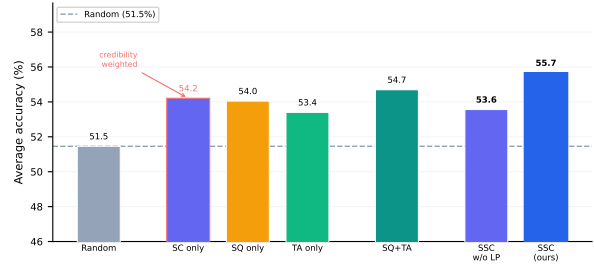


Figure 3: Controlled same-pool selector variants on the no-reference-answer pool (pooled, $n=618$). Combining weak signals does not necessarily yield monotonic improvements.

All variants reuse the same 6,180 candidates.

No single symbolic signal dominates across all benchmarks. The strongest single and pairwise variants reach 54.2% and 54.7%, respectively. The fixed three-signal fusion reaches 53.6%; this non-monotonicity shows that adding a weak signal can change close candidate rankings in either direction. Incorporating the two LP-based diagnostics increases pooled accuracy to 55.7%.

5.2 LP Diagnostics and Robustness

Objective proximity provides most of the observed LP gain, while relaxed-value profiles add a smaller complementary contribution. The result is insensitive to the profile dimension over the tested range: dimensions 10, 20, 30, 50, 100, and full padding all yield 55.7%, with 615–618 of 618 selected indices agreeing with dimension 30. Across the central LP-weight grid, accuracy remains stable, varying only from 55.3% to 55.7%. These are same-pool robustness checks, not evidence of independent out-of-sample tuning. Importantly, the LP relaxation remains only a weak diagnostic: sorting discards variable identity, and the resulting features certify neither integer feasibility nor semantic correctness.

5.3 Pool Coverage as a Diagnostic

Pool coverage provides a diagnostic ceiling on same-pool selection. It reaches 66.0% (408/618), meaning that 210 problems contain no answer-correct candidate and cannot be solved by any same-pool selector. Among the 408 reachable problems, SSC fails to select an answer-correct candidate in 64 cases. The resulting 10.4-percentage-point gap to the same-pool oracle indicates that further gains require improvements in both candidate generation and candidate selection.

Backbone	Method	NL4OPT	MaMO-E	MaMO-C	ComplexOR	IndOR	OptiBench	OptMath
DeepSeek-V3	Standard	70.5	84.3	39.8	52.6	29.0	52.4	16.2
DeepSeek-V3	CoT (Wei et al., 2022)	74.0	82.9	40.7	52.6	35.0	53.1	21.1
DeepSeek-V3	CoE (Xiao et al., 2024)	79.2	85.9	43.1	63.2	33.0	55.2	24.1
GPT-4o	Standard	70.3	84.3	41.2	57.8	27.0	42.3	17.5
GPT-4o	CoT (Wei et al., 2022)	71.6	84.8	42.3	57.8	29.0	42.0	20.5
GPT-4o	CoE (Xiao et al., 2024)	76.4	85.7	46.4	68.4	34.0	43.2	18.6
deepseek-v4-flash	SSC (ours, $K=10$)	80.0	92.0	39.0	55.6	34.0	43.0	46.0

Table 2: Published prompt-only results reported by OptiTree (Liu et al., 2025), shown together with our SSC result for context. The DeepSeek-V3/GPT-4o rows are single-pass generation results; SSC selects from our independently generated deepseek-v4-flash, $K=10$ pools. These results therefore provide contextual comparison rather than a controlled head-to-head evaluation.

Method	Accuracy (%)							
	NL4OPT	MaMO-E	MaMO-C	ComplexOR	IndOR	OptiBench	OptMath	All
Random draw (first position)	71.0	87.0	40.0	55.6	32.0	41.0	41.0	52.1
Random expected	73.4	88.3	37.4	50.6	32.1	38.4	39.3	51.5
Zero-shot LLM judge	76.0	88.0	41.0	55.6	35.0	40.0	42.0	53.7
LOBO learned ranker	78.0	92.0	39.0	55.6	32.0	43.0	48.0	55.3
SC only	77.0	90.0	37.0	55.6	33.0	41.0	47.0	54.2
SQ only	71.0	89.0	43.0	55.6	32.0	42.0	47.0	54.0
TA only	78.0	88.0	40.0	55.6	34.0	41.0	39.0	53.4
SSC w/o LP	80.0	90.0	36.0	55.6	34.0	41.0	40.0	53.6
SSC (ours) [†]	80.0	92.0	39.0	55.6	34.0	43.0	46.0	55.7
Pool coverage	83.0	94.0	51.0	55.6	42.0	53.0	75.0	66.0

Table 3: Controlled same-pool selection results for our generated candidate pools ($K=10$, $n=618$). Random expected averages the per-problem correct fraction. Random draw uses the shuffled first position and provides the paired baseline for significance tests. SSC w/o LP uses only SC, SQ, and TA; SSC (ours) adds LP-relaxation solution and objective consistency. Pool coverage is the fraction of problems with at least one correct candidate in the $K=10$ pool, not a selection method. [†] The significance marker refers only to the pooled McNemar test of SSC (ours) against Random draw (first position); $p = 0.0087$.

Variant	Accuracy (%)
Random expected	51.5
SC only	54.2
SQ only	54.0
TA only	53.4
SC+SQ	54.2
SC+TA	53.4
SQ+TA	54.7
SSC w/o LP	53.6
SSC (ours)	55.7

Table 4: Pooled component ablation. Fail rate denotes the proportion of coverage-reachable problems for which selection fails.

5.4 Signal Discriminability Analysis

We measure signal discriminability as the mean score on answer-correct candidates minus the mean on answer-incorrect candidates, pooling all ten positions within each benchmark. Table 6 relates these differences to the available selection headroom over the shuffled first-candidate baseline.

Variant	Accuracy (%)
Symbolic SSC	53.6
LP only	53.6
Symbolic + LP-obj	55.5
Symbolic + LP-sol	54.9
Full SSC	55.7

Table 5: Controlled LP-component ablation on the frozen no-reference-answer pool.

The most discriminative signal varies across benchmarks: TA on NL4OPT and MaMO-Complex, SQ on ComplexOR and OptMath, and SC on IndustryOR. OptiBench has weak deltas for all three symbolic signals and negative SC discriminability, consistent with its small observed selection gain. MaMO-Complex has positive TA and SQ deltas but near-zero SC and a one-point regression relative to first position, illustrating why pooled improvements should not be interpreted as uniform per-benchmark dominance.

Benchmark	Coverage gap	SSC gain	Gap recovered	Δ_{TA}	Δ_{SQ}	Δ_{SC}
NL4OPT	+12.0	+9.0	75.0%	+0.103	+0.004	+0.018
MaMO-Easy	+7.0	+5.0	71.4%	+0.005	+0.072	+0.072
MaMO-Complex	+11.0	-1.0	-9.1%	+0.063	+0.040	≈ 0
ComplexOR	0.0	0.0	-	+0.014	+0.160	+0.145
IndustryOR	+10.0	+2.0	20.0%	+0.043	+0.070	+0.194
OptiBench	+12.0	+2.0	16.7%	+0.002	+0.015	-0.021
OptMath	+34.0	+5.0	14.7%	+0.070	+0.235	+0.158

Table 6: No-reference-answer per-benchmark signal discriminability (Δ : correct-candidate mean minus incorrect-candidate mean) and coverage headroom. Bold marks the largest symbolic delta in each row.

Benchmark	No cand.	Sel. fail	Fail rate
NL4OPT	17	3	3.6%
MaMO-Easy	6	2	2.1%
MaMO-Complex	49	12	23.5%
ComplexOR	8	0	0.0%
IndustryOR	58	8	19.0%
OptiBench	47	10	18.9%
OptMath	25	29	38.7%
Total	210	64	15.7%

Table 7: Error decomposition. Fail rate denotes the proportion of coverage-reachable problems for which selection fails.

5.5 Error Analysis

We distinguish two types of incorrect SSC predictions: generation failures, where the candidate pool contains no answer-correct candidate, and selection failures, where at least one answer-correct candidate is available but SSC fails to select it.

The easiest benchmarks have high pool coverage and low reachable-error rates. OptMath presents the largest selection bottleneck: although 75 problems contain a correct candidate, SSC misses 29 of them. These failures motivate the development of numerically sensitive checks, richer relational-semantic signals, and stronger learned rerankers. Note that we do not reuse the submitted pool’s failure-cause labels for the replacement pool.

6 Related Work

NLP for Optimization. Early work on natural language to optimization focused on constrained problem templates (Ramamonjison et al., 2022). Recent systems generate solver-ready code using solver feedback (Ahmaditeshnizi et al., 2024; Liu et al., 2025) or domain-specific fine-tuning (Huang et al., 2025a; Jiang et al., 2025). We instead study MILP-solve-free selection from fixed candidate pools.

Self-Consistency and best-of- K . Self-consistency (Wang et al., 2023), best-of- N sam-

pling (Snell et al., 2024), and inference-time search (Yao et al., 2023) exploit multiple candidates. Program selection uses execution agreement or learned verification (Li et al., 2022; Chen et al., 2021; Shi et al., 2022; Chen et al., 2023; Ni et al., 2023), while non-execution reranking and functional-overlap clustering provide alternatives (Inala et al., 2022; Zhang et al., 2023; To et al., 2024). SSC combines MILP-specific structural, textual, and bounded LP-relaxation signals without integer optimization.

Prompting and representations. CoT (Wei et al., 2022) and CoE (Xiao et al., 2024) improve generation but do not directly address selection. AST similarity and type signatures have been used in code clone detection (Jiang et al., 2007); we adapt them to MILP-specific representations that capture mathematical structure rather than surface code.

7 Conclusion

We studied integer-solve-free selection of LLM-generated MILP programs. On fixed no-reference-answer pools, SSC achieves 55.7% accuracy, compared with a 51.5% random-selection expectation, 53.7% for a zero-shot LLM judge, and 55.3% for a LOBO learned ranker. Symbolic fusion alone achieves 53.6%, with LP-relaxation diagnostics contributing an additional 2.1 pp. Pool coverage reaches 66.0%, showing that a substantial fraction of failures originate from generation rather than selection alone. The remaining 10.4 pp gap to the same-pool oracle leaves meaningful room for improved selection. Together with the contextual published baselines, these results position SSC as a competitive and interpretable weak-verification baseline, rather than a universal reranker or correctness verifier. Remaining generation and selection errors motivate richer relational-semantic signals, numerically sensitive checks, and stronger learned evidence.

Limitations

All benchmarks require numeric reference answers for offline evaluation; open-ended problems are therefore excluded. We generate a single stochastic candidate pool per problem, so our results do not capture variation across independent generation seeds. The fixed weights were initially chosen heuristically and later adjusted with reference to results on the available pool rather than an independent development set; significance and robustness checks should therefore be read as descriptive same-pool evidence. Pool coverage further limits selector performance: 210 of 618 problems contain no answer-correct candidate. Finally, TA covers only coarse closed-vocabulary requirements, while sorted LP profiles discard variable identity and continuous relaxations cannot verify integrality or semantic correctness.

Ethics Statement

This work uses publicly available benchmark datasets (NL4OPT, MaMO, ComplexOR, IndustryOR, OptiBench, OptMath) and open-source solvers (SCIP (Achterberg, 2009), GLPK (Free Software Foundation, 2023)). No human subjects are involved. The LLM API calls required for generation and semantic requirement extraction carry a moderate compute cost; we mitigate this by caching semantic requirements and reusing them across ablation variants. We do not foresee direct ethical risks from MILP program selection, though any deployment of automated optimization modeling should be subject to domain-expert review before operational use.

Acknowledgments

The research is partially supported by Quantum Science and Technology–National Science and Technology Major Project (QNMP) 2021ZD0302900, China National Natural Science Foundation with No. 92567301, 62132018, 62231015, “Pioneer” and “Leading Goose” R&D Program of Zhejiang, 2023C01029 and 2023C01143, Anhui Provincial Science and Technology Innovation Program 202523o09050019 and 202523o09050015, and the USTC Kunpeng-Ascend Scientific and Educational Innovation Excellence Center.

References

- Tobias Achterberg. 2009. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41.
- Ali Ahmaditeshnizi, Wenzhi Gao, and Madeleine Udell. 2024. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. *arXiv preprint arXiv:2402.10172*.
- Michael L. Bynum, Gabriel A. Hackebeil, William E. Hart, Carl D. Laird, Bethany L. Nicholson, John D. Sirola, Jean-Paul Watson, and David L. Woodruff. 2021. *Pyomo—Optimization Modeling in Python*, third edition. Springer.
- Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. 2023. [CodeT: Code generation with generated tests](#). In *The Eleventh International Conference on Learning Representations*.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nick Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- Yinghao Chen, Wantong Xie, Shuli Zeng, Sijia Zhang, Xiaotian Pan, Feng Wu, and Xiangyang Li. 2026. [GOBench: Stage-wise diagnostics and the visual paradox in multimodal graph optimization](#). In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 6144–6167, San Diego, California, United States. Association for Computational Linguistics.
- DeepSeek-AI. 2024. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*.
- DeepSeek-AI. 2026. DeepSeek API change log. <https://api-docs.deepseek.com/updates/>. Accessed 2026-08-25.
- Free Software Foundation. 2023. GLPK (gnu linear programming kit). Version 5.0. <https://www.gnu.org/software/glpk/>.
- William E Hart, Carl Laird, Jean-Paul Watson, and David L Woodruff. 2011. Pyomo: modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, 3(3):219–260.
- Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. 2025a. [ORLM: A customizable framework in training large models for automated optimization modeling](#). *Operations Research*, 73(6):2986–3009.
- Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. 2025b. LLMs for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In *Findings of the*

- Association for Computational Linguistics: NAACL 2025, pages 2678–2710, Albuquerque, New Mexico. Association for Computational Linguistics.
- Jeevana Priya Inala, Chenglong Wang, Mei Yang, Andres Codas, Mark Encarnación, Shuvendu K. Lahiri, Madanlal Musuvathi, and Jianfeng Gao. 2022. Fault-aware neural code rankers. In *Advances in Neural Information Processing Systems*, volume 35.
- Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. 2025. LLMOPT: Learning to define and solve general optimization problems from scratch. In *The Thirteenth International Conference on Learning Representations*.
- Lingxiao Jiang, Ghassan Mishserghi, Zhendong Su, and Stephane Glondou. 2007. DECKARD: Scalable and accurate tree-based detection of code clones. In *Proceedings of the 29th International Conference on Software Engineering*, pages 96–105. IEEE Computer Society.
- Qingyang Li, Lele Zhang, and Vicky Mak-Hau. 2023. Synthesizing mixed-integer linear programming models from natural language descriptions. *arXiv preprint arXiv:2311.15271*.
- Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, and 1 others. 2022. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097.
- Haoyang Liu, Jie Wang, Yuyang Cai, Xiongwei Han, Yufei Kuang, and Jianye Hao. 2025. OptiTree: Hierarchical thoughts generation with tree search for LLM optimization modeling. In *Advances in Neural Information Processing Systems*.
- Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. 2025. OptMATH: A scalable bidirectional data synthesis framework for optimization modeling. In *Proceedings of the 42nd International Conference on Machine Learning*, pages 40769–40802. PMLR.
- Quinn McNemar. 1947. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157.
- George L Nemhauser and Laurence A Wolsey. 1988. *Integer and combinatorial optimization*, volume 18. Wiley New York.
- Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-Tau Yih, Sida Wang, and Xi Victoria Lin. 2023. **LEVER: Learning to verify language-to-code generation with execution**. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 26106–26128. PMLR.
- Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Hadrien Cambazard, Bernard Gendron, and 1 others. 2022. NL4OPT competition: Formulating optimization problems based on their natural language descriptions. In *Proceedings of the NeurIPS 2022 Competitions Track*, pages 189–203. PMLR.
- Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I. Wang. 2022. **Natural language to code translation with execution**. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3533–3546. Association for Computational Linguistics.
- Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.
- Hung Quoc To, Minh Huynh Nguyen, and Nghi D. Q. Bui. 2024. **Functional overlap reranking for neural code generation**. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3686–3704. Association for Computational Linguistics.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35.
- Laurence A. Wolsey. 1998. *Integer Programming*. Wiley.
- Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. 2024. Chain-of-experts: When LLMs meet complex operations research problems. In *The Twelfth International Conference on Learning Representations*.
- Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhi-jiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Lingi Song, Xiaodan Liang, and Jing Tang. 2025. OptiBench meets ReSocratic: Measure and improve LLMs for optimization modeling. In *The Thirteenth International Conference on Learning Representations*.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*.
- Tianyi Zhang, Tao Yu, Tatsunori Hashimoto, Mike Lewis, Wen-tau Yih, Daniel Fried, and Sida I. Wang. 2023. Coder reviewer reranking for code generation. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 41832–41846. PMLR.

A SSC Algorithm and Key Implementation Conventions

Selector pseudocode. Given problem text $\mathcal{P}$ and candidates $\{c_i\}_{i=1}^K$:

1. construct each candidate in an isolated subprocess and extract its standard linear representation, with AST checks and fallbacks;
2. extract and cache one closed-vocabulary requirement set from $\mathcal{P}$; compute SQ and TA for each candidate;
3. compute pairwise structural similarities, clone components at threshold 0.95, credibility-weighted SC, and the symbolic base;
4. cap the symbolic base by SQ when a fatal static or construction defect is present;
5. solve each continuous relaxation under the bounded GLPK protocol and pool-normalize LP-sol and LP-obj;
6. fuse the terms in Eq. 9 and return the highest-scoring candidate, using stable candidate order for ties.

No reference answer, offline correctness label, or integer-solver output is available to these steps.

Exact LP diagnostics. Let $v_i = P_{30}(\text{sort}_{\downarrow}(\tilde{x}_i))$, where P_{30} truncates or zero-pads to 30 entries. Let $\mathcal{F}$ index valid, nonempty LP profiles and $\mathcal{F}_i = \mathcal{F} \setminus \{i\}$. With $\cos_+(a, b) = \max(0, \cos(a, b))$,

$$r_i^{\text{sol}} = \begin{cases} \frac{\sum_{j \in \mathcal{F}_i} \cos_+(v_i, v_j)}{|\mathcal{F}_i|}, & i \in \mathcal{F}, \mathcal{F}_i \neq \emptyset, \\ 0, & \text{otherwise.} \end{cases} \quad (15)$$

Zero-norm pairs contribute zero. Let $\mathcal{F}_z$ index candidates with a feasible objective; let m be the upper middle order statistic used by the implementation and let $R = \max_j \tilde{z}_j - \min_j \tilde{z}_j$, replacing R by 1 below 10^{-9} :

$$r_i^{\text{obj}} = \begin{cases} \max(0, 1 - |\tilde{z}_i - m|/R), & i \in \mathcal{F}_z, \\ 0, & \text{otherwise.} \end{cases} \quad (16)$$

Each diagnostic is divided by its positive pool maximum, or set to zero when that maximum is zero.

Key prompt contracts. Generation is instructed not to include, infer, or use a gold/reference answer and to return a `pyo.ConcreteModel` from `build_model(data=None)`. The five formulation-emphasis variants ask for natural named components, data-first construction, compact indexed models with direction checks, decomposed constraints, or an audit-first plan. They are paired

with temperatures 0.6, 0.8, 1.0, 1.2, and 1.4 and repeated twice. The semantic-extraction prompt returns JSON from a 21-feature vocabulary and is instructed not to solve the problem or use a reference answer.

B Benchmark Details

Table 1 summarises the seven evaluation benchmarks. The Easy/Medium/Hard labels are descriptive rather than official benchmark categories; they reflect a combination of benchmark design, formulation complexity, and observed first-position accuracy and pool-coverage behaviour in our $K=10$ candidate pools.

NL4OPT. NL4OPT (Ramamonjison et al., 2022) was introduced as a competition benchmark for formulating linear programs from natural-language business descriptions. Problems are drawn from production planning, logistics, and resource-allocation contexts. All problems are pure LPs (no integer or binary variables), and their descriptions are relatively short and structurally regular. We evaluate the first 100 records of the normalized test file (230 problems in the full split). We label NL4OPT **Easy** because first-position accuracy is high (71%) and pool coverage reaches 83%. TA is the dominant discriminative signal ($\Delta_{\text{TA}} = +0.103$), consistent with the structured, vocabulary-rich nature of the descriptions.

MaMO-Easy. MaMO (Huang et al., 2025b) was introduced at NAACL 2025 as a benchmark for mathematical modeling with LLMs, covering a broad range of combinatorial and continuous optimization problems described in natural and mathematical language. MaMO-Easy is the easier subset of MaMO, with 652 problems in total; we use the first 100 normalized test records. Problems have clearer structure and fewer secondary conditions than the Complex subset. We label MaMO-Easy **Easy** due to high pool coverage (94%) and high first-position accuracy (87%). LP-relaxation signals contribute a small improvement (+2 pp), serving as tie-breakers when symbolic signals are saturated.

MaMO-Complex. MaMO-Complex is the harder subset of MaMO (Huang et al., 2025b) (211 problems; we use the first 100 normalized test records). Problems have longer descriptions, more secondary conditions, multi-index constraints, and often require mixed-integer formulations. We label

MaMO-Complex Medium: pool coverage is 51%, indicating that generation quality is a significant bottleneck, but selection failures are also substantial (23.5% of coverage-reachable problems). LP signals improve the symbolic selector by 3 points (36%→39%), and SC discriminability is near zero ($\Delta_{SC} \approx 0$).

ComplexOR. ComplexOR (Xiao et al., 2024) was introduced as a benchmark of complex operations-research formulations, including multi-commodity flow, vehicle routing, facility location, and scheduling variants. All 18 problems are used (no sampling required). We label ComplexOR **Hard** because problems are long and structurally demanding, with many binary and integer decision variables. SSC achieves 55.6%, matching pool coverage; no selector can improve on this particular pool. SQ has the largest symbolic delta ($\Delta_{SQ} = +0.160$).

IndustryOR. IndustryOR (Huang et al., 2025a) was introduced with the ORLM framework as a benchmark of industry-style optimization problems drawn from supply-chain planning, production scheduling, and facility management. All 100 problems are used. We label IndustryOR **Hard** because real-world problem descriptions are often heterogeneous and underspecified, making consistent formulation difficult. Pool coverage is only 42%, placing a hard ceiling on any selector. SSC achieves 34% accuracy (+2 pp over first), illustrating a signal-pool mismatch: the global Δ_{SC} is relatively high (+0.194) but does not reliably identify the correct candidate within individual pools.

OptiBench. OptiBench (Yang et al., 2025) includes data augmented via the ReSocratic procedure, covering a broad range of LP and MILP problem types from academic and application domains. We use the first 100 normalized test records (605 problems in the full split). We label OptiBench **Hard** due to problem diversity and limited pool coverage (53%). SSC reaches 43% (+2 pp over first); all three symbolic deltas are weak and SC is slightly negative ($\Delta_{SC} = -0.021$).

OptMath. OptMath (Lu et al., 2025) uses a bidirectional data synthesis framework, with problems derived from mathematical competition problems and scaled to various difficulty levels. We use the first 100 normalized test records (166 problems in the full split). We label OptMath **Hard** and it exhibits the largest selection headroom of any

benchmark in our evaluation: first-position accuracy is 41%, while pool coverage reaches 75%, giving a 34 pp gap. SSC reaches 46% (+5 pp over first); LP relaxation adds 6 points over the symbolic fusion, and SQ is the strongest symbolic signal ($\Delta_{SQ} = +0.235$).

C Published Prompt-Only Baselines

The published Standard, CoT, and CoE rows are shown in the main paper (Table 2) because baseline comparison is a primary experimental objective. OptiTree reports these results using DeepSeek-V3 and GPT-4o. Because they differ from our setup in model backbone, $K = 10$ candidate pool, and selection budget, they provide contextual rather than controlled comparisons.

D No-Reference-Answer Error Counts

Table 7 reports the complete generation-versus-selection error decomposition for the replacement pool. We retain only counts that can be recomputed from the frozen evaluation labels; the more subjective submitted-pool failure-cause annotations are not transferred to this pool.

E Similarity Function Details

E.1 Variable Similarity

Each variable signature v stores a domain label d , index arity $a \in \mathbb{Z}_{\geq 0}$, bound expression b , and raw token set τ . Variable similarity is:

$$s_v(v, v') = 0.35 \cdot \mathbf{1}[d=d'] + 0.30 \cdot g(a, a') + 0.15 \cdot J(b, b') + 0.20 \cdot J(\tau, \tau'). \quad (17)$$

where $g(a, a') = \max(0, 1 - |a - a'|/3)$ and J is the set Jaccard coefficient. For the domain term, matching an unknown domain to a known domain receives a score of 0.4 rather than 0. Pairwise matching uses greedy best-match alignment, normalized by the size of the larger collection.

E.2 Constraint Similarity

The frozen selection path compares constraint collections directly rather than greedily matching individual rows. Let q_n compare collection sizes with h , and let q_s , q_a , q_t , q_v , and q_p compare, respectively, the bags of canonical senses, arities, linear terms, variable shapes, and the set of parameter tokens aggregated over all constraint rows. The

runtime constraint-collection similarity is

$$s_c = 0.20q_n + 0.15q_s + 0.10q_a + 0.35q_t + 0.15q_v + 0.05q_p. \quad (18)$$

E.3 Objective Similarity

For objectives, let s_e be bag Jaccard over the combined canonical semantic tokens (magnitude-binned numbers, operators, variable shapes, linear terms, and parameter tokens). Let s_p denote the weighted combination of the corresponding expression-profile similarities with weights 0.08 (numbers), 0.10 (operators), 0.18 (variable shapes), 0.17 (parameters), 0.37 (linear terms), and 0.10 (signed numbers). Objective similarity is then:

$$s_o = 0.35s_{\text{direction}} + 0.25s_e + 0.40s_p. \quad (19)$$

For $s_{\text{direction}}$, a direction mismatch receives 0, while an unknown direction receives 0.2. If both objectives are absent, the objective similarity is 1.0; if exactly one is absent, it is 0.

E.4 Parameter-Dependency Similarity

Parameter-dependency similarity is a lightweight auxiliary term over tokens aggregated from the full model. Let α, α' be parameter-token sets and let ν, ν' be numeric-token multisets. We define

$$s_d = 0.75 \cdot J(\alpha, \alpha') + 0.25 \cdot J_{\text{bag}}(\nu, \nu'), \quad (20)$$

where J and J_{bag} are set and bag Jaccard similarity, respectively. Canonical linear representations intentionally contain no parameter-name tokens, so the first term is active only for AST fallback representations; this auxiliary score is therefore primarily numeric on successfully standardized candidates.

E.5 Shape Similarity

Shape similarity captures global structural statistics:

$$\begin{aligned} s_s = & 0.25 \cdot h(|\mathcal{V}|, |\mathcal{V}'|) + 0.25 \cdot h(|\mathcal{R}|, |\mathcal{R}'|) \\ & + 0.20 \cdot \mathbf{1}[\text{has_obj} = \text{has_obj}'] \\ & + 0.15 \cdot J_{\text{bag}}(\text{domains}) \\ & + 0.15 \cdot J_{\text{bag}}(\text{senses}). \end{aligned} \quad (21)$$

where $h(a, b) = \max(0, 1 - |a - b| / \max(1, \max(a, b)))$.

E.6 Overall Program Similarity

The five sub-scores are combined with fixed weights:

$$\begin{aligned} \text{sim}(\mathcal{M}_i, \mathcal{M}_j) = & 0.18 \cdot s_v + 0.27 \cdot s_o + 0.45 \cdot s_c \\ & + 0.05 \cdot s_d + 0.05 \cdot s_s, \end{aligned} \quad (22)$$

where s_v, s_o, s_c, s_d, s_s are the variable, objective, constraint, parameter-dependency, and shape sub-scores defined above. Weights reflect the relative informativeness of each structural layer: constraints receive the largest weight (0.45), followed by the objective (0.27) and variable signatures (0.18); shape and parameter-dependency serve as lightweight regularizers (0.05 each).

F Static Quality Defect Taxonomy

Table 8 lists the defect types and penalty weights used in Eq. 13. Defects flagged as fatal additionally cap the symbolic base before LP fusion: $B_i \leftarrow \min(B_i^{\text{raw}}, \text{SQ}(c_i))$.

G Full Per-Benchmark Results

Table 9 reports the full per-benchmark results for First, SSC, and pool coverage, used in all statistical analyses.

H Clone-Aware Consistency Weighting

When multiple candidates are near-identical, a naive mean of pairwise similarities inflates the consistency score of the dominant cluster. We address this with a union-find clone grouping: two candidates c_i, c_j are placed in the same clone group when $\text{sim}(\mathcal{M}_i, \mathcal{M}_j) \geq \tau_{\text{clone}} = 0.95$. Each candidate receives $w_i = \text{SQ}(c_i) \text{TA}(c_i, \mathcal{P}) / |\text{clone-group}(i)|$, and the weighted consistency score is:

$$\text{SC}(c_i) = \frac{\sum_{j \neq i} w_j \text{sim}(\mathcal{M}_i, \mathcal{M}_j)}{\sum_{j \neq i} w_j}. \quad (23)$$

Thus clone diversity and candidate credibility affect SC jointly. The threshold $\tau_{\text{clone}} = 0.95$ is fixed across all benchmarks.

I Per-Benchmark Ablation Results

Table 10 gives the full ablation breakdown by benchmark for all component variants. Each entry reports accuracy (%).

Category	Defect type	Penalty
Parse / AST	syntax_error	1.00 [†]
	missing_variables	0.30 [†]
	missing_objective	0.30 [†]
	missing_constraints	0.25
	unknown_objective_direction	0.08
	unknown_variable_domain	0.08
	unknown_constraint_sense	0.06
	possible_nonlinearity	0.15
Pyomo API	reserved_pyomo_component_name	0.60 [†]
	rule_arity_mismatch	0.50 [†]
	unsupported_pyomo_max	0.45 [†]
	unsupported_python_max_on_pyomo_expr	0.10
	disallowed_component_decorator	0.00 [†]
	disallowed_constraint_list	0.00 [†]
	disallowed_add_call	0.00 [†]
	disallowed_dynamic_component	0.00 [†]
	disallowed_lambda_rule	0.00
	contains_solver_call	0.15 [†]
Sandbox	sandbox_import_error	0.70 [†]
	sandbox_construct_error	0.70 [†]
	sandbox_timeout	0.70 [†]
	sandbox_no_model	0.65 [†]
	sandbox_no_variables	0.45 [†]
	sandbox_no_objective	0.45 [†]
	sandbox_no_constraints	0.35 [†]
	sandbox_subprocess_error	0.60 [†]
	sandbox_bad_json	0.60 [†]
Standardization	standardization_import_error	0.70 [†]
	standardization_construct_error	0.70 [†]
	standardization_no_model	0.65 [†]
	standardization_no_variables	0.45 [†]
	standardization_no_objective	0.45 [†]
	standardization_no_constraints	0.35 [†]
	standardization_timeout	0.70 [†]
	standardization_subprocess_error	0.70 [†]
	standardization_bad_json	0.60 [†]

Table 8: Static quality defect taxonomy. [†] marks fatal defects that cap the symbolic base before LP fusion; a zero penalty can still trigger this cap when the construct violates the code contract.

J Statistical Significance and Confidence Intervals

J.1 McNemar’s Test

We use two-sided McNemar’s test (McNemar, 1947) to assess whether SSC differs from the first-position random draw on each benchmark. Since candidate positions are shuffled independently per problem (Appendix M.5), the first candidate is a valid random draw while still enabling paired testing. Let b denote the number of problems where the first-position draw is correct and SSC is wrong; c denote the number of problems where the first-position draw is wrong and SSC is correct. For $b + c < 25$ we use the exact binomial; otherwise the chi-squared statistic with continuity correction. Pooling all seven benchmarks gives $b=21$ and $c=43$ discordant pairs; the continuity-corrected McNemar statistic is $\chi^2=6.891$ with $p=0.0087$. Because design choices were informed by results on this pool, we interpret this result as descriptive

paired evidence rather than an independent confirmatory test.

K Controlled Neural Baselines and Selection Cost

The zero-shot listwise judge sees the problem and all ten candidate programs in a deterministically shuffled order. Reference-answer assignments are masked, and neither solver outputs nor offline labels are provided. A single deepseek-v4-flash pass per problem selects 332/618 candidates correctly (53.72%). The run uses 5,684,192 input tokens and 55,545 output tokens. Against SSC, there are 38 SSC-only and 26 judge-only successes (paired exact $p=0.169$).

The LOBO ranker is an L2-regularized logistic regression over the same stored SSC component features. It trains on candidate-level correctness labels from six benchmarks and tests on the held-out seventh, obtaining 342/618 (55.34%). SSC-only and ranker-only successes are 6 and 4 (paired exact $p=0.754$). Note that fixed SSC uses no task-specific training labels.

L Semantic Requirement Extraction

Text alignment (Section 3.3) relies on LLM-extracted semantic requirements from the problem description $\mathcal{P}$. We prompt an LLM to return a JSON list of requirements, each with fields: name (short identifier), features (one or more tokens from a closed vocabulary), and weight (importance, $\in [0.1, 2.0]$). Extraction uses a DeepSeek-family chat endpoint at temperature 0, with a 1400-token cap and two retries. The resulting requirement set depends only on the problem text; it is cached in `semantic_requirements.json` and reused for all K candidates and all ablations. The prompt contains only the problem statement and the closed feature vocabulary, not candidate code, solver outputs, or gold answers, thus TA has no access to evaluation labels. If the response is invalid JSON or no supported vocabulary feature remains after filtering, requirement extraction is considered unsuccessful and the experiment terminates.

The closed vocabulary of 21 semantic features is:

Method	NL4OPT	MaMO-E	MaMO-C	CpxOR	IndOR	OptiBench	OptMath
First	71.0	87.0	40.0	55.6	32.0	41.0	41.0
SSC	80.0	92.0	39.0	55.6	34.0	43.0	46.0
Coverage	83.0	94.0	51.0	55.6	42.0	53.0	75.0

Table 9: Per-benchmark accuracy used for statistical analysis ($K=10$, $n=618$).

Variant	NL4OPT	MaMO-E	MaMO-C	ComplexOR	IndOR	OptiBench	OptMath	Avg
Random	73.4	88.3	37.4	50.6	32.1	38.4	39.3	51.5
SC only	77.0	90.0	37.0	55.6	33.0	41.0	47.0	54.2
SQ only	71.0	89.0	43.0	55.6	32.0	42.0	47.0	54.0
TA only	78.0	88.0	40.0	55.6	34.0	41.0	39.0	53.4
SC + SQ	77.0	90.0	37.0	55.6	33.0	41.0	47.0	54.2
SC + TA	80.0	89.0	36.0	55.6	34.0	41.0	40.0	53.4
SQ + TA	78.0	89.0	41.0	55.6	34.0	42.0	44.0	54.7
SSC w/o LP	80.0	90.0	36.0	55.6	34.0	41.0	40.0	53.6
SSC (ours)	80.0	92.0	39.0	55.6	34.0	43.0	46.0	55.7

Table 10: Per-benchmark ablation accuracy (%). SSC w/o LP uses Eq. 8; full SSC uses Eq. 9.

Benchmark	n	b	c	p -value	
NL4OPT	100	1	10	0.0117	*
MaMO-Easy	100	1	6	0.1250	n.s.
MaMO-Complex	100	5	4	1.0000	n.s.
ComplexOR	18	0	0	1.0000	n.s.
IndustryOR	100	3	5	0.7266	n.s.
OptiBench	100	2	4	0.6875	n.s.
OptMath	100	9	14	0.4049	n.s.
Pooled	618	21	43	0.0087	**

Table 11: McNemar’s test for SSC vs. the first-position random draw ($K=10$).

Vars	Constr.	Time (s)	Peak MiB
10	10	0.157	0.4
100	100	0.688	2.5
500	500	3.267	12.3
1000	1000	6.395	24.2

Table 12: Controlled symbolic SSC cost for $K = 10$ synthetic linear Pyomo candidates. Time includes construction, canonicalization, and symbolic scoring; it excludes generation and LP solving.

objective:minimize	objective:maximize
var:binary	var:integer
var:bounded	constraint:equality
constraint:equality_binary	constraint:equality_continuous
constraint:upper_bound	constraint:lower_bound
constraint:assignment	constraint:capacity
constraint:budget	constraint:resource
constraint:demand	constraint:coverage
constraint:linking	constraint:facility_linking
constraint:flow_balance	constraint:routing
constraint:inventory_balance	

Requirements with feature tokens outside this vocabulary are discarded. Weights for objective sense are typically 1.5; structural constraint types (assignment, flow balance) are 1.0–1.5; bound-type constraints are 0.5–1.0.

Model-side evidence is extracted from the candidate’s parsed representation without LLM involvement: variable domains yield `var:binary`, `var:integer`, `var:bounded`; objective sense yields `objective:minimize/maximize`; constraint senses and expression profiles are matched to semantic constraint types via rule-based pattern matching. The TA score is the weighted fraction of requirements whose evidence features appear in the candidate’s evidence set.

M Generation Pipeline Details

M.1 Template Retrieval

We maintain a library of 14 canonical MILP problem archetypes (e.g., Transportation, Assignment, Knapsack, Facility Location, Network Flow, Scheduling). Each template records: problem type name, natural language description, canonical formulation sketch, reasoning flow steps, and common modeling errors.

For a problem description $\mathcal{P}$, we score each template by keyword overlap:

$$\begin{aligned} \text{score}(t) = & \sum_{k \in K_t} (5 + |k|) \mathbf{1}[k \subseteq \mathcal{P}] \\ & + 1.25 \sum_{k \in K_t} |\text{words}(k) \cap \text{words}(\mathcal{P})| \\ & + 0.15 |Q \cap T|, \end{aligned} \quad (24)$$

where K_t is the keyword set for template t , and Q and T are the word sets of $\mathcal{P}$ and the full template text. Templates with $\text{score} \geq \max_t \text{score}(t) \times 0.5$

and score ≥ 5 are retrieved (up to three); otherwise a generic “General LP/MILP” template is used. Retrieved templates are appended to the generation prompt as a retrieval-augmented context block.

M.2 Formulation Variants

Each of the $K = 10$ candidates is generated with a distinct formulation emphasis instruction appended to the prompt (candidate_index mod 5, cycling through the five variants twice):

1. **Natural naming:** use descriptive variable names (e.g., production[i], assign[i, j]); one Pyomo Constraint component per semantic restriction.
2. **Data-first:** enumerate every numeric value from the problem in DATA with its source sentence before defining any Pyomo component; reference only DATA entries in the model.
3. **Compact index style:** use short variable names (x, y, z) with explicit RangeSet; verify each constraint direction ($\leq$ vs. $\geq$) against the problem text after writing it.
4. **Decomposed:** one Pyomo Constraint rule per atomic sentence; introduce auxiliary binary variables for any if-then logic or indicator linearization.
5. **Audit-first:** write a structured comment block listing all decision variables (with index sets and domains), the objective in words, and each constraint numbered to its source sentence; then translate directly to Pyomo and verify sense/directions a second time.

M.3 CoT Reasoning Steps

Every generation prompt includes six reasoning steps that the LLM is instructed to work through before writing code:

1. **Sets and indices:** list every collection of entities (machines, products, time periods, ...).
2. **Variables:** for each decision, name the variable, its index set, and its domain (Binary / Integer / NonNegativeReals / Reals).
3. **Objective:** write the objective expression in words, then confirm minimize vs. maximize.
4. **Constraints:** enumerate every restriction from the problem text; check that each sentence maps to at least one constraint.

5. **Data usage:** confirm every numeric value in the problem appears in DATA and is referenced in the model.

6. **Linearity:** ensure no products of two decision variables appear in any expression.

M.4 Temperature Schedule and Repair

The ten candidates are generated using a cycled temperature schedule of [0.6, 0.8, 1.0, 1.2, 1.4, 0.6, 0.8, 1.0, 1.2, 1.4] (the five-value conservative-to-exploratory spectrum repeated twice). This schedule promotes candidate diversity while avoiding the near-deterministic outputs that lower temperatures (e.g., 0.2) can produce for the same prompt.

After each initial generation, a non-optimization repair loop runs for up to two rounds. A candidate triggers repair when any of the following is detected:

- Syntax error or Pyomo API violation (reserved name, rule arity mismatch, disallowed ConstraintList / add() usage)
- Sandbox construction failure (import error, construct error, timeout)
- Missing Pyomo components (no variables, no objective, no constraints)
- Possible nonlinearity or solver call

Repair prompts include the original formulation variant instruction, the retrieved template, the problem text, and structured feedback from the sandbox and AST checks. Repairs use temperature 0 to encourage deterministic corrections while preserving the mathematical intent of the original formulation.

M.5 Random-Draw Baseline

Random expected. The expected accuracy of a uniform random draw from the $K=10$ candidate pool, computed as the mean per-problem correct fraction:

$$\text{Random}_{\text{exp}} = \frac{1}{n} \sum_{p=1}^n \frac{1}{K} \sum_{k=1}^K \mathbf{1}[\text{correct}(c_k^p)]. \quad (25)$$

Random draw (first position). The accuracy of always selecting the first candidate in the shuffled pool:

$$\text{Random}_{\text{first}} = \frac{1}{n} \sum_{p=1}^n \mathbf{1}[\text{correct}(c_1^p)]. \quad (26)$$

Why the two values differ. Both are unbiased estimates of the same underlying quantity, but $\text{Random}_{\text{exp}}$ averages all K positions per problem and therefore has lower finite-sample variance, while $\text{Random}_{\text{first}}$ is a single-position sample (0 or 1 per problem) with higher variance. On finite benchmarks the two can differ noticeably—for example, MaMO-Easy: 87.0% vs. 88.3%; ComplexOR: 55.6% vs. 50.6% (Table 3). We report $\text{Random}_{\text{first}}$ as the paired baseline for McNemar’s test (Appendix J) because it gives one concrete decision per problem.

Why first position is a valid random draw. Each problem’s ten generation recipes—the five formulation-emphasis and temperature pairs, each repeated twice—are shuffled with a problem-specific seed `f"{seed}:{problem_id}"` before being assigned to candidate positions 0–9. Every position therefore receives each recipe with equal probability $\frac{1}{10}$ across problems, so c_1^p is statistically equivalent to a uniform random draw.

The practical consequence is that the random expected accuracy (51.5%) is the expected baseline for any position-agnostic selection method that ignores candidate content.

M.6 Pyomo Code Contract

All generated candidates must satisfy the following structural contract:

- Use a concrete model: `model = pyo.ConcreteModel()`.
- All Pyomo components defined as `model.<name> = pyo.Var/Constraint/...`
- Objective: `model.obj = pyo.Objective(expr=..., sense=...)`
- Indexed constraints via named rule functions; not `ConstraintList`, `add()`, lambda rules, or dynamic component names in loops.
- No `SolverFactory`, `.solve()`, or `pyo.value()` calls inside `build_model()`.
- DATA dictionary as the single source of numeric constants.
- If a compatibility placeholder is needed, define `GOLD_ANSWER = None`; never use a reference answer in the model.

LP dimension	Correct	Agree with 30
10	344/618	615/618
20	344/618	618/618
30	344/618	618/618
50	344/618	616/618
100	344/618	616/618
Full padded	344/618	616/618

Table 13: Sensitivity to the sorted LP-solution profile dimension.

N LP Robustness Checks

Table 13 varies the sorted LP-profile dimension while keeping the candidate pools and all other scores fixed. All tested dimensions produce the same answer accuracy; a small number of selected candidate identities differ without affecting correctness.

A cached grid over the central LP blending weights yields 55.3–55.7% accuracy. These checks show local same-pool stability, but they do not replace independent hyperparameter validation. We omit the submitted lambda plot and the approximate prefix- K curve because they did not rerun the complete pool-dependent selector under each displayed configuration.

O Frozen Experimental Provenance

The replacement experiment contains 618 problems and 6,180 candidates, exactly ten candidates per problem. All seven July 11 manifests record the rolling API endpoint string `deepseek-v4-flash`, thinking disabled, seed 0, and no generation-time diversity conditioning. The cached TA requirements date from May 18, and the listwise judge was run on August 25; the shared endpoint name does not assert checkpoint identity across these dates. Candidate order is shuffled deterministically by problem and seed. Python 3.11.15, Pyomo 6.10.0, NumPy 2.4.3, SciPy 1.17.1, and GLPK 5.0 were recorded in the audited environment; SCIP evaluation uses a 120-second process timeout. The implementation is available at <https://github.com/Faramita2/SSC-MILP>.