AUTOMATED INVERSE OPTIMIZATION WITH LLMS VIA VERIFICATION-GUIDED HYPOTHESIS SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Inverse optimization (IO) infers latent decision-making models from observed decisions, but classically requires experts to rigidly specify the objective, constraints, and behavioral assumptions prior to estimate model parameters. While recent advances in large language models (LLMs) have enabled automated construction of optimization models from natural language, they struggle in the inverse setting, where the latent model structure must be iteratively inferred from behavioral demonstrations. We formulate automated IO as a structure-aware learning problem, jointly identifying the objective components, latent constraints, behavioral assumptions, and numerical parameters from observed decisions. To address this, we propose VeriHy, an LLM-based framework that performs verification-guided search over competing model hypotheses. VeriHy organizes hypotheses in a directed acyclic graph and uses evidence from successful candidates and counterexamples to guide model refinement and recombination. Experiments across convex, combinatorial, and real-world IO problems show that VeriHy achieves performance competitive with expert-crafted classical methods and significantly outperforms LLM baselines under matched search budgets.

1 INTRODUCTION

Inverse optimization (IO) reverses the usual optimization problem: rather than computing decisions from a known objective, it infers latent costs, preferences, or parameters from observed decisions (Ahuja & Orlin, 2001; Keshavarz et al., 2011; Chan et al., 2025). Modern IO methods handle noisy or imperfect observations and learn decision models from repeated demonstrations (Aswani et al., 2018; Mohajerin Esfahani et al., 2018; Bärmann et al., 2017). A common assumption, however, is that the *structural form* of the forward optimization model is known in advance. The inverse problem then estimates unknown coefficients within this predefined model family.

Recent work has relaxed this assumption by recovering constraint parameters or learning feasible regions from demonstrations (Chan & Kaw, 2020; Ghobadi & Mahmoudzadeh, 2021; Ren et al., 2025; Kitaoka, 2025). Nevertheless, these approaches still operate within prescribed model classes. In practice, the model itself may be misspecified: an objective component may be missing, an optional constraint may be incorrect, or the assumed behavioral mechanism may not match the demonstrator. In such cases, improved parameter estimation alone cannot resolve the structural error.

Large language models (LLMs) provide a natural mechanism for exploring such structural alternatives. Recent systems can translate natural-language descriptions into optimization models, generate solver code, and iteratively repair formulations (Xiao et al., 2024; Ahmaditeshnizi et al., 2024; Zhang & Luo, 2025). Astorga et al. (Astorga et al., 2025) further cast autoformulation as search over competing mathematical formulations. Verification-based approaches additionally use structural or solver feedback to identify incorrect models (Lian et al., 2026; Liu et al., 2026; Li et al., 2026; Li & Hai, 2026). These methods primarily address a *specification-to-model* problem, where an intended problem description is available. Our setting is different: the specification itself is latent, and candidate models must instead be inferred from observed decisions.

We therefore introduce VERIHY, a verification-guided framework for structural inverse optimization. A model hypothesis is represented by its objective components, active constraints, and behavioral assumption. The LLM proposes structural changes, while deterministic inverse-optimization

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.

routines estimate the numerical parameters of each candidate. The fitted model is then independently evaluated on held-out demonstrations. Verification failures are summarized as behavioral evidence and used to guide subsequent revisions.

VeriHy organizes evaluated hypotheses in a directed acyclic graph (DAG), preserving their scores, counterexamples, and structural relationships. This enables the search to refine promising hypotheses, avoid repeated failures, and combine complementary components from different branches. Importantly, every new structure is refitted and reverified; parameters are never inherited directly from its parents.

We evaluate VeriHy across convex, combinatorial, and real-world inverse-optimization problems. In addition to held-out decision quality, we study search efficiency, structural recovery, and sensitivity to the proposal LLM. Our contributions are:

- We formulate inverse optimization under *structural uncertainty*, where objective components, constraints, and behavioral assumptions may be unknown in addition to their numerical parameters.
- We propose VERIHY, which combines LLM-based structural hypothesis generation with deterministic inverse fitting and verification-guided search over a hypothesis DAG.
- We evaluate VeriHy on convex and combinatorial benchmarks and real-world last-mile routing, measuring both predictive performance and recovery of the latent model structure.

2 RELATED WORK

Inverse optimization. Classical IO estimates objective or preference parameters that rationalize observed decisions (Ahuja & Orlin, 2001; Keshavarz et al., 2011). Subsequent work addresses noisy, imperfect, and repeated observations (Aswani et al., 2018; Mohajerin Esfahani et al., 2018; Bärmann et al., 2017); see Chan et al. (2025) for a recent overview. Most of this literature assumes a specified forward-model family. Closer to our setting, several works infer unknown constraint parameters or feasible regions (Chan & Kaw, 2020; Ghobadi & Mahmoudzadeh, 2021; Ren et al., 2025), while Kitaoka (2025) considers learning both constraints and objectives within a prescribed mixed-integer model class. VeriHy instead searches over discrete alternatives in the model structure and delegates numerical estimation within each candidate to conventional IO procedures.

LLMs for optimization modeling. LLMs have increasingly been used to formulate optimization problems from textual specifications. Chain-of-Experts decomposes complex operations-research modeling across specialized reasoning roles (Xiao et al., 2024), while OptiMUS and OR-LLM-Agent combine formulation, solver-code generation, execution, and repair (Ahmaditeshnizi et al., 2024; Zhang & Luo, 2025). Most closely related, Astorga et al. (2025) search over alternative mathematical formulations using Monte Carlo tree search. These methods address a forward *specification-to-model* setting. VeriHy instead considers a *behavior-to-model* setting: the optimization specification is unknown, and candidate structures are evaluated after fitting their parameters to demonstrations.

Verification and search. Recent systems strengthen LLM-based optimization with explicit verification. ReLoop performs behavioral verification, Opt-Verifier combines structural and solution-level checks, and OptArgus and falsification-based methods detect incorrect or inconsistent formulations (Lian et al., 2026; Liu et al., 2026; Li et al., 2026; Li & Hai, 2026). VeriHy uses verification for a different purpose: failures are treated as evidence for revising an unknown decision model rather than checking conformance to a given specification. The resulting search maintains a DAG of fitted hypotheses and uses behavioral counterexamples to guide structural refinement and fusion.

3 PROBLEM FORMULATION

We consider inverse optimization from demonstrations $\mathcal{D} = \{(u_i, y_i)\}_{i=1}^{N}$, where u_i is the observed context and y_i the corresponding expert decision. Unlike classical inverse optimization, which assumes a known forward model and estimates only its parameters, we aim to recover both the numerical parameters and the latent decision structure.

We represent a model hypothesis as

$$H = (\Phi, G, R), \tag{1}$$

where $\Phi \subseteq \mathcal{L}_{\text{obj}}$ is a subset of candidate objective components, $G \subseteq \mathcal{L}_{\text{con}}$ contains the active constraints, including all mandatory base constraints, and $R \in \mathcal{R}$ denotes a behavioral assumption such as exact or noisy expert optimization. Each hypothesis induces

$$x^\star_H(u;\theta) \in \arg\max_{x \in \mathcal{X}_G(u)} f_\Phi(x,u;\theta), \tag{2}$$

$$f_\Phi(x,u;\theta) = f_0(x,u) + \sum_{k\in\Phi} \theta_k^\top \psi_k(x,u), \tag{3}$$

where ψ_k may be nonlinear in (x,u) but is linear in its parameter block θ_k, and $\mathcal{X}_G(u)$ is defined by the constraints in G. This block-linear form supports linear and quadratic objectives, group-specific coefficients, and derived decision features while preserving tractable inverse fitting.

The data are split into training, validation, audit, and test sets. Parameters are estimated on $\mathcal{D}_{\text{tr}}$, hypotheses are selected using $\mathcal{D}_{\text{val}}$, top candidates are checked once on $\mathcal{D}_{\text{audit}}$, and the test set is reserved for final evaluation. The learning problem is

$$H^\star \in \arg\min_{H\in\mathcal{H}} J_{\text{val}}\Big(H, \widehat{\theta}(H;\mathcal{D}_{\text{tr}})\Big), \tag{4}$$

where $\widehat{\theta}$ is obtained by inverse-optimization fitting and $\mathcal{H}$ is a combinatorial space over objective terms, constraints, and behavioral models. Since exhaustive enumeration is infeasible, VeriHy-IO constructs a directed DAG of hypotheses and expands it using validation counterexamples.

4 METHOD

We study inverse optimization when both the model structure and its numerical parameters are unknown. Our framework, VERIHY, searches over candidate models proposed by an LLM, but delegates parameter estimation and evaluation to deterministic optimization routines. Figure 1 summarizes the loop: propose a hypothesis, check its validity, fit its parameters, verify its predictions, and use the resulting counterexamples to guide the next proposal.

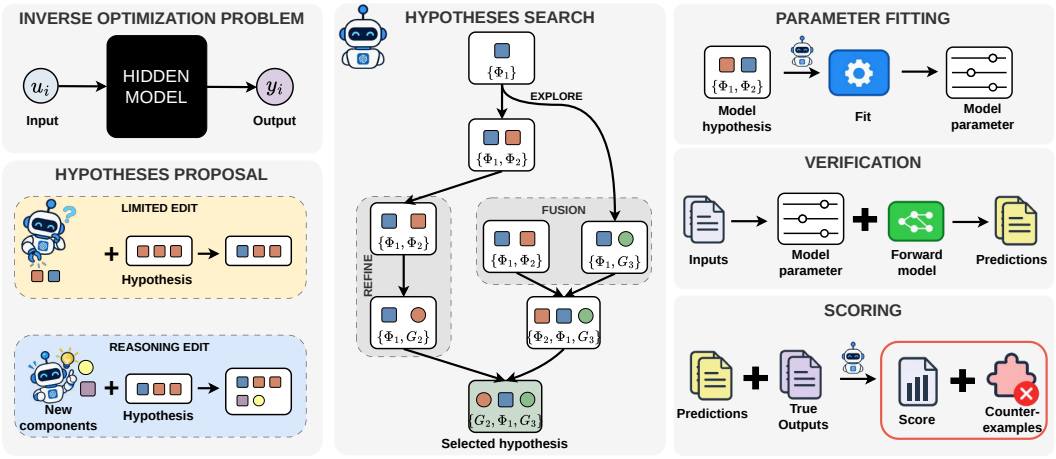

Figure 1: Overview of VeriHy. The LLM proposes hypotheses for structural edits to the problems. Validation counterexamples are used to guide parent selection for fusion, refine or subsequent proposals in the hypothesis search graph. The final hypothesis is selected by validation score.

4.1 HYPOTHESES AND LLM PROPOSAL SCOPE

Given demonstrations $\mathcal{D} = \{(u_i, y_i)\}_{i=1}^N$, we seek a forward model that explains how the observed context u_i leads to the decision y_i. Rather than assuming that the model structure is known, VeriHy-IO represents each candidate as a hypothesis $H = (\Phi, G, R)$: Φ selects objective components,

G selects constraints, and R describes the expert's behavior (e.g., exact or noisy optimization). It induces

$$x_H^\star(u;\theta) \in \arg\max_{x \in \mathcal{X}_G(u)} f_\Phi(x,u;\theta),$$
$$f_\Phi(x,u;\theta) = f_0(x,u) + \sum_{k\in\Phi} \theta_k^\top \psi_k(x,u). \tag{5}$$

The mandatory constraints are included in every G; ψ_k may be nonlinear in (x,u) but is linear in its parameter block θ_k. The problem specification supplies component libraries, a decision loss, and a forward solver. We search over the discrete structure H while fitting its numerical parameters on training data:

$$\widehat{H} \in \arg\min_{H\in\mathcal{H}} J_{\mathrm{val}}\left(H, \underbrace{\mathrm{Fit}_R(H,\mathcal{D}_{\mathrm{tr}})}_{\widehat{\theta}_H}\right). \tag{6}$$

The search is approximate because $\mathcal{H}$ grows combinatorially with candidate components. This separation fixes the LLM's role: it proposes structural changes, whereas fitting and forward solving are performed by compiled optimization routines.

The degree of LLM intervention follows the task's validated interface. M0 ranks admissible edits deterministically from verification evidence; M1 lets the LLM select an edit from that menu; M2 lets it compose one or two admissible edits; and M3 lets it propose a new objective or constraint component when the task can compile and validate new components. Thus a benchmark with a closed, complete library can use M1/M2, while an incomplete library with an executable component interface can additionally use M3. The LLM sees the task specification, candidate edits, and summaries of previously evaluated hypotheses, but never the latent structure or test set. All proposals undergo the same syntax, feasibility, solver-compatibility, and empirical-identifiability checks; see Appendix B.

4.2 Node Evaluation

4.2.1 Parameter fitting.

For each admitted H, we fit $\widehat{\theta}_H$ using only $\mathcal{D}_{\mathrm{tr}}$. The behavioral assumption R and compiled problem properties select the fitter. To illustrate, for exact discrete choices we seek parameters that prefer each demonstration y_i over every feasible alternative z:

$$\max_{\theta,\rho,\xi} \quad \rho - C\sum_{i\in I_{\mathrm{tr}}} \xi_i - \lambda_\theta \|\theta\|_1 \tag{7}$$
$$\text{s.t.} \quad f_\Phi(y_i,u_i;\theta) - f_\Phi(z,u_i;\theta) \ge \rho - \xi_i,$$
$$i \in I_{\mathrm{tr}}, \quad z \in \mathcal{X}_G(u_i)\setminus\{y_i\},$$
$$\xi_i \ge 0, \quad \theta \in \Theta_\Phi. \tag{8}$$

The shared margin $\rho \in [0, \rho_{\max}]$ and slack ξ_i allow weak or imperfectly separated demonstrations. A cutting-plane oracle finds violating alternatives instead of enumerating them. For noisy discrete choices, we use loss-augmented structured margins; for suitable continuous problems, we minimize a KKT residual. We reject unsupported model–fitter pairs. Appendix C gives the full objectives, oracles, and assumptions.

4.2.2 Independent verification.

We freeze $\widehat{\theta}_H$ and solve the candidate forward model on held-out validation contexts. Its prediction $\widehat{x}_i^H$ can disagree with an expert decision even when the training fit is good. The primary validation loss and the full node score are

$$L_{\mathrm{dec}}(H) = \frac{1}{|I_{\mathrm{val}}|}\sum_{i\in I_{\mathrm{val}}} \Delta(\widehat{x}_i^H, y_i), \tag{9}$$
$$J(H) = L_{\mathrm{dec}}(H) + \lambda_{\mathrm{gap}} L_{\mathrm{gap}}(H) + \lambda_{\mathrm{feas}} L_{\mathrm{feas}}(H) + \lambda_{\mathrm{unc}} U(H) + \lambda_{\mathrm{cmp}} C(H). \tag{10}$$

Here Δ is the task's decision loss. L_{gap} measures the candidate's objective advantage for its prediction over a feasible demonstration; L_{feas} penalizes constraints contradicted by demonstrations or empty feasible sets. U measures variation in parameter fits across training folds, and C penalizes unnecessary components. For example, if $\widehat{x}_i^H \neq y_i$ but their objective values are similar, decision loss detects disagreement while the gap can remain small. All score terms use the candidate and observed data, never the ground-truth model. Appendix D specifies their normalization and edge cases.

4.2.3 COUNTEREXAMPLE EVIDENCE.

A score ranks nodes, but does not indicate what to change. We therefore retain the validation counterexamples and summarize which missing model components might explain them:

$$\begin{aligned} \mathcal{C}_H &= \{i : \Delta(\widehat{x}_i^H, y_i) > \tau_\Delta\}, \\ \bar{d}_{k,g}(H) &= \frac{1}{|\mathcal{C}_H|} \sum_{i \in \mathcal{C}_H} \big[\psi_{k,g}(y_i, u_i) - \psi_{k,g}(\widehat{x}_i^H, u_i)\big], \\ \pi_k(H) &= \max_g |\bar{d}_{k,g}(H)|/s_{k,g}, \qquad k \notin \Phi. \end{aligned} \tag{11}$$

The fixed probe scale $s_{k,g}$ makes feature residuals comparable; taking the signed mean before the absolute value prevents conflicting errors from looking consistently informative. For constraints, we test whether they exclude observed decisions (v_c) or eliminate wrong predictions while retaining the observations (e_c). Together with the strength a_k of active objective blocks, these statistics form the evidence summary $E_H = (\{\pi_k\}, \{a_k\}, \{v_c\}, \{e_c\})$. It guides additions, removals, and constraint swaps, but each edit must be refitted and reverified. Exact definitions and the empty-counterexample convention are in Appendix D.

4.3 EVIDENCE-GUIDED SEARCH OVER HYPOTHESIS DAG

The search DAG stores a node $v = (H_v, \widehat{\theta}_v, J_v, E_v)$ for every distinct admitted hypothesis. An edge $v \xrightarrow{a} v'$ records the edit a and parent that produced v'. At iteration t, we first sample an anchor parent from expandable nodes according to

$$p_t(v) \propto \exp\big(-\widetilde{J}_v/T\big), \tag{12}$$

where $\widetilde{J}_v$ is a normalized validation score. When search stagnates, additional parents receive more weight for behavioral diversity, measured by disagreement between their validation predictions rather than by textual differences between model descriptions:

$$D_{\text{beh}}(v, w) = \frac{1}{|I_{\text{val}}|} \sum_{i \in I_{\text{val}}} \Delta(\widehat{x}_i^v, \widehat{x}_i^w). \tag{13}$$

Selected parents pass their scores, counterexamples, and tried edits to the proposal policy. It may add or remove an objective term, add, remove, or swap an optional constraint, or change R. With complementary parents, it can also propose a fused child that imports objective components (e.g., $\Phi' \subseteq \Phi_v \cup \Phi_w$). Fusion still passes admission, fitting, and verification; parameters are never copied from the parents.

In each iteration we generate candidates $\mathcal{A}_t = \pi_M(\mathcal{S}, \mathcal{P}_t, \{(J_p, E_p)\}_{p \in \mathcal{P}_t})$, where $\mathcal{P}_t$ is the selected parent set. For each admissible novel child $H' = a(\mathcal{P}_t)$, we compute $(\widehat{\theta}_{H'}, J(H'), E_{H'})$ and add its provenance edges to the DAG. Duplicate or empirically equivalent proposals share a representative; invalid proposals receive no fit. We stop at a fixed evaluation or compute budget (or when no expandable node remains) and return the node with the lowest validation score. A fixed top-K shortlist may be checked once on a separate audit split without changing that choice; test data are reserved for final evaluation. Initialization, fusion, equivalence tests, and the full iteration are in Appendix E.

5 EXPERIMENTS

We evaluate VeriHy on convex, combinatorial, and real-world inverse-optimization problems. In every experiment, model parameters are fitted on demonstrations, structural hypotheses are selected

using validation data, and the test set is reserved for final evaluation. Our study addresses four research questions:

- **RQ1: Decision quality and search efficiency.** How does VeriHy compare with classical inverse optimization (IO) methods and state-of-the-art LLM-based search frameworks in terms of decision quality and search efficiency across convex and combinatorial benchmarks?
- **RQ2: Search dynamics and LLM choice.** How does VeriHy perform both at the final search budget and throughout the search process? How does the choice of proposal LLM affect its performance?
- **RQ3: Structural recovery.** How accurately does VeriHy recover the objective components and latent constraints underlying the observed demonstrations?
- **RQ4: Real-world routing.** Can VeriHy learn an effective decision model from historical routes in the Amazon Last Mile Routing Research Challenge?

5.1 Experimental Setup

Convex IO. Five synthetic IO families spanning different objectives and feasible-set geometries follow the same settings from Li et al. (Li et al., 2025). We select *Simplex-IO* and *Additive-Box-IO*: these are among the more challenging settings in our preliminary evaluation, while their coupled simplex decisions and coordinate-wise box decisions yield behaviorally distinct test cases. Contexts and latent objective parameters generate optimal decisions under the forward model. In the clean setting, these decisions are observed directly; in the noisy setting, the demonstrations are perturbed. We use 50 and 100 training examples and report test regret under the generating objective. Results for the remaining convex families, together with data generation and solver details, are deferred to Appendix A.1.

Combinatorial inverse optimization. We consider knapsack, scheduling, and bipartite matching with 50, 100, and 200 demonstrations, follow the same setting from Navarro et al. (Navarro et al., 2026). Candidate models contain alternative objective components and, for scheduling and constrained matching, alternative latent constraints such as precedence relations or forbidden edges. Recovering the generating model therefore requires explaining both decision quality and feasibility. We report penalized test regret in percent; infeasible predictions and solver failures receive a 100% instance-level penalty. Because the generating model is known, we additionally measure objective-component and constraint recovery. Appendix A.2 gives the complete feature libraries, hidden structures, data-generation procedures, and forward solvers.

Real-world routing. We use the Amazon Last Mile Routing Research Challenge data, which contains historical stop sequences and route-, stop-, and package-level information from operational delivery routes (Merchán et al., 2022). Following the routing IO formulation of Scroccaro et al. (2024), a route is represented as a sequence over delivery zones and stops, and the learner infers costs that reproduce expert-driver preferences rather than merely minimizing travel time. Performance is measured by the official route-discrepancy score, for which lower is better. Appendix A.3 describes preprocessing, the candidate objective components, the train/test protocol, and score computation.

Comparators. For convex problems, we compare with the classical KKA, MOM, and ERM estimators used by Li et al. (Li et al., 2025). Across the automated-search comparisons, we include EoH (Liu et al., 2024), Opt-Verifier (Liu et al., 2026), MCTS-AHD (Zheng et al., 2025), Opt-Master (Lin et al., 2026), OpenEvolve (Sharma, 2025), and ShinkaEvolve (Lange et al., 2026). Every method receives the same task interface and is evaluated on the same held-out instances within a benchmark. Unless otherwise stated, LLM-based methods use `gpt-4o-mini` (OpenAI, 2024). For sensitivity analysis, we also run VeriHy with `gpt-5-nano` (OpenAI, 2025), `DeepSeek-V3.2` (DeepSeek-AI, 2025), and `Qwen3.5-35B-A3B` (Qwen Team, 2026).

5.2 Overall Comparison

On convex benchmarks (Table 1), VeriHy records the lowest displayed regret in six of eight settings. Opt-Verifier is best on the two clean simplex settings, whereas VeriHy performs best on both noisy

Table 1: Results on Convex Inverse Optimization benchmarks. The evaluation metric is test regret; lower is better. C denotes clean observations and N denotes noisy observations.

Method	Simplex IO				Additive Box IO			
	50-C	50-N	100-C	100-N	50-C	50-N	100-C	100-N
KKA	0.0272	0.0856	0.0102	0.0689	0.0168	0.2729	0.0054	0.2222
MOM	0.0184	0.0520	0.0023	0.0190	0.0111	0.0196	0.0037	0.0031
ERM	0.3163	0.5590	0.2366	0.2205	0.3057	0.4008	0.1735	0.2231
EoH	0.0852	0.0835	0.0860	0.1111	0.2517	0.4437	0.3687	0.2186
Opt-Verifier	**0.0042**	0.0253	**0.0021**	0.0048	0.0088	0.0115	0.0025	0.0129
MCTS-AHD	0.0995	0.0969	0.0798	0.0351	0.0422	0.5349	0.2459	0.1469
OptMaster	0.0140	0.0110	0.0074	0.0046	0.0110	0.0114	0.0024	0.0027
OpenEvolve	0.1751	0.1896	0.1033	0.1839	0.9371	0.8243	0.9385	0.6149
ShinkaEvolve	0.1580	0.1259	0.1011	0.1602	0.7141	0.4476	0.8936	0.3463
VeriHy	0.0046	**0.0063**	0.0040	**0.0016**	**0.0055**	**0.0059**	**0.0015**	**0.0014**

simplex settings and all additive-box settings. In particular, on the noisy additive-box setting at size 100, VeriHy achieves a regret of 0.0014 compared with 0.0129 for Opt-Verifier. These comparisons use the table's reported values.

Table 2: Results on Combinatorial Inverse Optimization benchmarks. The evaluation metric is penalized test regret (%); lower is better.

Method	Knapsack IO			Scheduling IO			Matching IO		
	50	100	200	50	100	200	50	100	200
EoH	2.840	2.480	0.060	27.280	26.770	2.330	26.510	1.200	1.900
Opt-Verifier	2.840	27.400	54.510	75.590	75.610	75.560	51.020	50.780	26.510
MCTS-AHD	0.120	0.090	0.030	26.740	21.890	21.480	75.050	1.840	50.970
OptMaster	75.020	25.080	72.590	75.570	95.860	51.160	100.000	100.000	51.110
OpenEvolve	0.170	2.190	2.350	70.460	100.000	75.590	2.090	2.090	1.730
ShinkaEvolve	25.160	0.060	75.000	89.250	50.580	100.000	51.110	26.650	50.970
VeriHy	**0.100**	**0.040**	**0.020**	**0.760**	**0.610**	**1.070**	**0.950**	**0.650**	**1.170**

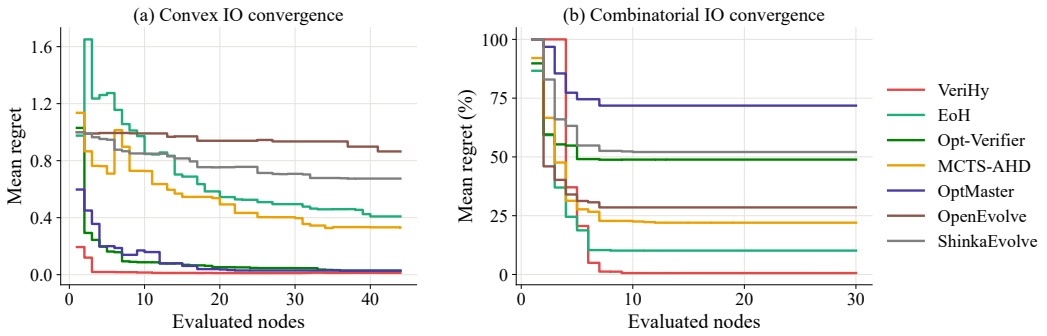

Figure 2: Aggregated convergence on the convex and combinatorial benchmarks. Lower is better.

In Table 2, VeriHy stays below 1.5% penalized regret in every combinatorial setting. It achieves the lowest reported value in eight of nine columns. Several baselines incur large penalties when their predictions are infeasible or cannot be solved, most clearly on scheduling and the larger knapsack settings. Averaging the recorded nine settings, VeriHy reaches 0.60% penalized regret and produces feasible predictions on all tested instances.

Figure 2 complements the final-budget tables with an anytime comparison. On convex tasks, VeriHy's normalized test regret falls to 0.019 by node 5 and 0.013 by node 44, compared with 0.029 for Opt-Verifier at node 44. On combinatorial tasks, VeriHy reaches 0.82% average penalized regret by node 10. Measured by LLM usage, it reaches 0.82% combinatorial regret at 10k cumulative tokens; on the convex tasks, it reaches 0.015 normalized regret at 25k tokens, compared with 0.079 for Opt-Verifier. Because each curve reports the test performance of the validation-selected hypothesis at a search prefix, test regret need not decrease monotonically.

5.3 Search Efficiency and LLM Choice

Search traces give an anytime view of the validation-selected hypothesis. On convex tasks, its test regret normalized by the zero-parameter reference falls to 0.019 by node 5 and 0.013 by node 44, compared with Opt-Verifier's 0.029 at node 44. On combinatorial tasks, VeriHy reaches 0.82% average penalized regret by node 10. These curves evaluate the hypothesis selected using validation at each search prefix; therefore held-out regret can fluctuate even when validation performance improves.

At a cumulative 10k LLM tokens, VeriHy reaches 0.82% combinatorial regret. Over the convex settings, normalized test regret is 0.015 at 25k tokens, compared with 0.079 for Opt-Verifier. We report these prefix results as a measure of search efficiency, without treating token thresholds as separately tuned training budgets.

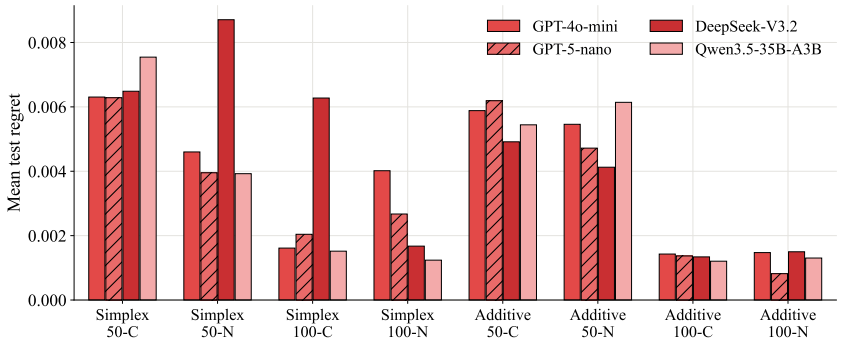

Figure 3: Different LLM choices effects on convex IO problems.

Figure 3 compares VeriHy with four LLM backbones on the convex IO benchmarks. The impact of the backbone is instance-dependent, with larger variations on Simplex 50-N and Simplex 100-C and smaller differences on several 100-dimensional Additive settings. No model consistently achieves the lowest regret across all cases, suggesting that VeriHy remains effective across different proposal models while retaining some sensitivity to the LLM choice.

5.4 Structural Recovery

For each combinatorial benchmark, the generating objective-component set $\Phi^\star$ and constraint set $G^\star$ are known. For each combinatorial benchmark, the generating objective-component set $\Phi^\star$ and constraint set $G^\star$ are known. We count a run as an exact structural recovery only when the recovered objective components exactly match $\Phi^\star$ and the recovered non-base constraints exactly match $G^\star$. Base constraints shared by all hypotheses are excluded from this comparison. To capture partial objective recovery, we additionally report Objective F1:

$$F_1 = \frac{2PR}{P+R}, \qquad P = \frac{|\hat{\Phi} \cap \Phi^\star|}{|\hat{\Phi}|}, \quad R = \frac{|\hat{\Phi} \cap \Phi^\star|}{|\Phi^\star|}.$$

Finally, feasible rate is the fraction of test decisions produced by the recovered model that satisfy the true constraints $G^\star$.

Figure 4 shows that VeriHy achieves 91.7% exact structural recovery and 100% recovery of the true constraints. It also reaches 95% objective-component F1 and 100% test feasibility. In comparison,

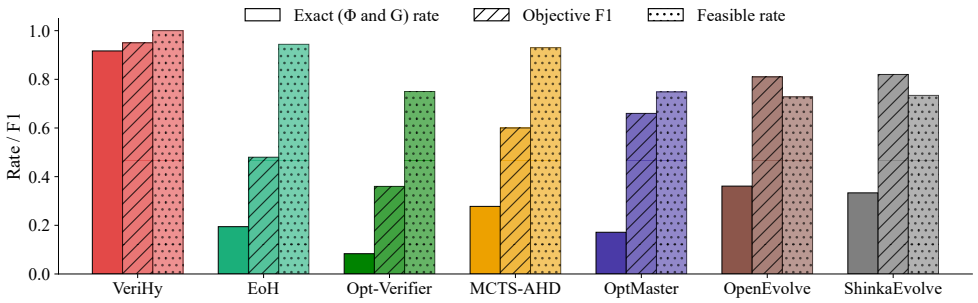

Figure 4: Structural recovery on the combinatorial benchmarks. Higher is better.

the strongest alternative in exact structural recovery, OpenEvolve, achieves 36.1%. These results indicate that VeriHy more consistently recovers the generating structure while preserving feasibility under the true constraints.

5.5 Real-World IO Problems

We compare VeriHy against Just Passing Through (JPT) (Cook et al., 2024), the winning solution of the Amazon Last Mile Routing Research Challenge, as well as Contextual Routing (Sun & Arslan, 2025), Routing IO (Scroccaro et al., 2024), and Two-stage GRASP (Mesa et al., 2023).

Table 3: Amazon Last Mile results under the official route-discrepancy metric; lower is better.

Method	**JPT**	**VeriHy (Claude Code)**	**Contextual Routing**	**Routing IO**	**Two-stage**
Route score ↓	**0.0248**	0.0265	0.0294	0.0302	0.0328

Real-world evaluation. VeriHy is coupled with Claude Code (Anthropic, 2025) to generate routing hypotheses, implement candidate algorithms, and revise them using verification feedback. As shown in Table 3, VeriHy achieves a route score of 0.0265, outperforming the other data-driven baselines and approaching the challenge-winning result.

6 Conclusions

We introduced VeriHy, a verification-guided framework for automated inverse optimization under structural uncertainty. By combining LLM-based hypothesis generation with deterministic fitting and verification, VeriHy searches over objective components, constraints, and behavioral assumptions rather than assuming a fixed model structure. Experiments across convex, combinatorial, and real-world problems demonstrate the potential of this approach for recovering useful decision models from demonstrations.

A key limitation is computational efficiency: each hypothesis still requires inverse fitting and verification, which may become costly for larger hypothesis spaces and more challenging learning-augmented problems. Future work should improve the scalability of this search process and extend it to settings involving richer machine-learning components. More broadly, we hope this work highlights the potential of LLMs as hypothesis generators for the automated design and discovery of optimization models.

REFERENCES

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

Ravindra K. Ahuja and James B. Orlin. Inverse optimization. *Operations Research*, 49(5):771–783, 2001. doi: 10.1287/opre.49.5.771.10607.

Anthropic. Claude 3.7 sonnet and claude code, 2025. URL https://www.anthropic.com/news/claude-3-7-sonnet.

Nicolás Astorga, Tennison Liu, Yuanzhang Xiao, and Mihaela Van Der Schaar. Autoformulation of mathematical optimization models using LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 1864–1886. PMLR, 2025.

Anil Aswani, Zuo-Jun Shen, and Auyon Siddiq. Inverse optimization with noisy data. *Operations Research*, 66(3):870–892, 2018. doi: 10.1287/opre.2017.1705.

Andreas Bärmann, Sebastian Pokutta, and Oskar Schneider. Emulating the expert: Inverse optimization through online learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 400–410. PMLR, 2017.

Timothy C. Y. Chan and Neal Kaw. Inverse optimization for the recovery of constraint parameters. *European Journal of Operational Research*, 282(2):415–427, 2020. doi: 10.1016/j.ejor.2019.09.027.

Timothy C. Y. Chan, Rafid Mahmood, and Ian Yihang Zhu. Inverse optimization: Theory and applications. *Operations Research*, 73(2):1046–1074, 2025. doi: 10.1287/opre.2022.0382.

William Cook, Stephan Held, and Keld Helsgaun. Constrained local search for last-mile routing. *Transportation Science*, 58(1):12–26, 2024. doi: 10.1287/trsc.2022.1185.

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models, 2025. URL https://arxiv.org/abs/2512.02556.

Kimia Ghobadi and Houra Mahmoudzadeh. Inferring linear feasible regions using inverse optimization. *European Journal of Operational Research*, 290(3):829–843, 2021. doi: 10.1016/j.ejor.2020.08.048.

Arezou Keshavarz, Yang Wang, and Stephen Boyd. Imputing a convex objective function. In *2011 IEEE International Symposium on Intelligent Control*, pp. 613–619, 2011.

Akira Kitaoka. Inverse mixed-integer programming: Learning constraints then objective functions, 2025. URL https://arxiv.org/abs/2510.04455.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=lKEdGCoDNC.

Haifeng Li and Mo Hai. Falsification-based verification of LLM-generated optimization models: Sound test batteries and their detection limits, 2026.

Zhehao Li, Yanchen Wu, Jian Chen, and Xiaojie Mao. A fenchel-young loss approach to data-driven inverse optimization, 2025. URL https://arxiv.org/abs/2502.16120.

Zhong Li, Zihan Guo, Xiaohan Lu, Juntao Wang, Jie Song, Chao Shen, Jiageng Wu, and Mingyang Sun. OptArgus: A multi-agent system to detect hallucinations in LLM-based optimization modeling, 2026.

Junbo Jacob Lian, Yujun Sun, Huiling Chen, Chaoyu Zhang, and Chung-Piaw Teo. ReLoop: Structured modeling and behavioral verification for reliable LLM-based optimization, 2026.

Hang Lin, Yuanpeng Gao, Yuzhi Zhang, Kun Yuan, Gang Yan, Siheng Chen, Linfeng Zhang, and Weinan E. Optmaster: A DAG-based framework for formulation and heuristic discovery in optimization. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=zvbuhAxwKy.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32201–32223. PMLR, 2024. URL https://openreview.net/forum?id=BwAkaxqiLB.

Haoyang Liu, Jie Wang, Boxuan Niu, Xiongwei Han, Yian Xu, Mingxuan Ye, Zijie Geng, Fangzhou Zhu, Tao Zhong, Mingxuan Yuan, and Jianye Hao. Opt-Verifier: Unleashing the power of LLMs for optimization modeling via dual-side verification, 2026.

Daniel Merchán, Jatin Arora, Julian Pachon, Karthik Konduri, Matthias Winkenbach, Steven Parks, and Joseph Noszek. 2021 amazon last mile routing research challenge: Data set. *Transportation Science*, 2022. doi: 10.1287/trsc.2022.1173.

Juan Pablo Mesa, Alejandro Montoya, Raúl Ramos-Pollán, and Mauricio Toro. A two-stage data-driven metaheuristic to predict last-mile delivery route sequences. *Engineering Applications of Artificial Intelligence*, 125:106653, 2023. doi: 10.1016/j.engappai.2023.106653.

Peyman Mohajerin Esfahani, Soroosh Shafieezadeh-Abadeh, Grani A. Hanasusanto, and Daniel Kuhn. Data-driven inverse optimization with imperfect information. *Mathematical Programming*, 167(1):191–234, 2018. doi: 10.1007/s10107-017-1216-6.

Macarena Navarro, Willem-Jan van Hoeve, and Karan Singh. Inverse optimization without inverse optimization: Direct solution prediction with transformer models, 2026. URL https://arxiv.org/abs/2602.05306.

OpenAI. GPT-4o mini: Advancing cost-efficient intelligence, 2024. URL https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/.

OpenAI. GPT-5 nano model documentation, 2025. URL https://developers.openai.com/api/docs/models/gpt-5-nano.

Qwen Team. Qwen3.5-35B-A3B model card, 2026. URL https://huggingface.co/Qwen/Qwen3.5-35B-A3B.

Ke Ren, Peyman Mohajerin Esfahani, and Angelos Georghiou. Inverse optimization via learning feasible regions. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 51471–51488. PMLR, 2025.

Pedro Zattoni Scroccaro, Piet van Beek, Peyman Mohajerin Esfahani, and Bilge Atasoy. Inverse optimization for routing problems. *Transportation Science*, 2024. doi: 10.1287/trsc.2023.0241.

Asankhaya Sharma. OpenEvolve: An open-source evolutionary coding agent, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve.

Huai Jun Sun and Okan Arslan. A contextual framework for learning routing experiences in last-mile delivery. *Transportation Research Part B: Methodological*, 194:103172, 2025. doi: 10.1016/j.trb.2025.103172.

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.

Bowen Zhang and Pengcheng Luo. OR-LLM-Agent: Automating modeling and solving of operations research optimization problem with reasoning large language model, 2025.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in LLM-based automatic heuristic design. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2501.08603.

A BENCHMARKS

We evaluate the proposed inverse-optimization framework on a collection of continuous and combinatorial benchmarks designed to test different aspects of inverse learning. The benchmarks vary along three main dimensions: the type of decision space, the complexity of the latent objective, and whether the feasible-set structure is completely known or must also be inferred.

For each problem instance, a context u_i is generated synthetically and an expert decision y_i is obtained by solving the corresponding forward optimization problem under a ground-truth objective and, when applicable, a ground-truth constraint structure. The inverse learner observes only the contexts and expert decisions and attempts to recover the latent structure. Ground-truth objective components, parameters, and hidden constraints are kept separate from the learner and are used only for evaluation.

The datasets are partitioned into training, validation, audit, and test sets. The training split is used to estimate objective parameters for candidate models, while the validation split is used to compare and select candidate structures. The audit split provides an additional out-of-sample check of the top-ranked hypotheses, and the test split is reserved exclusively for final evaluation. This separation is particularly important for the structural benchmarks, where the learner searches over multiple objective and constraint hypotheses.

A.1 CONVEX INVERSE OPTIMIZATION

The convex benchmarks isolate objective-parameter recovery in a controlled continuous setting. The feasible region is assumed to be fully known, and the unknown object is a parameter vector $\theta^\star \in \mathbb{R}^p$. Given context $u \in \mathbb{R}^p$, the expert decision is

$$y = x^\star(u;\theta^\star) \in \arg\max_{x\in\mathcal{X}(u)} f(x,u;\theta^\star). \tag{14}$$

The inverse learner receives demonstrations $(u_i, y_i)_{i=1}^N$ and estimates a parameter $\hat{\theta}$ whose induced forward decisions reproduce the observed expert behavior.

In all convex experiments, we use $p = 3$. Context vectors are generated as

$$u_i \sim \mathcal{N}(0, I_p), \tag{15}$$

while the latent coefficient vector is sampled independently from

$$\theta^\star \sim \mathcal{U}([-1,1]^p). \tag{16}$$

The five convex geometries share the same inverse-learning interface but differ in the geometry of their feasible sets and in how the context interacts with the latent parameter.

Simplex-IO. The Simplex-IO benchmark considers the forward problem

$$\max_{x\in\mathbb{R}^p} (\theta+u)^\top x \qquad \text{s.t.} \qquad x \succeq 0, \qquad \mathbf{1}^\top x \le 1. \tag{17}$$

Because the objective is linear, the optimizer places all available mass on a coordinate attaining the largest positive value of $\theta_j + u_j$. If every coordinate has non-positive utility, the zero vector is optimal. The benchmark therefore tests whether the inverse method can infer the latent preference vector $\theta^\star$ from changes in the active coordinate across contexts.

The forward oracle is available in closed form, making this benchmark useful for verifying the correctness of the inverse-learning implementation without introducing numerical errors from a generic mathematical-programming solver. Because different parameter vectors may induce the same sequence of active coordinates, direct parameter recovery is not always identifiable; behavioral metrics such as decision error and regret are therefore more informative than raw parameter error in this setting.

Additive-Box-IO. The Additive-Box-IO benchmark uses the hypercube $\mathcal{X} = [-1,1]^p$ and solves

$$\max_{x\in[-1,1]^p} (\theta+u)^\top x. \tag{18}$$

The optimal solution is coordinate-wise,

$$x_j^\star = \mathrm{sign}(\theta_j + u_j), \tag{19}$$

up to the specified tie-breaking convention when a coefficient is zero.

This benchmark tests additive interaction between an unobserved preference vector and an observed contextual signal. In contrast to the simplex geometry, all coordinates can simultaneously contribute to the decision, so each demonstration potentially reveals information about multiple entries of $\theta^\star$.

Hadamard-Box-IO. The Hadamard-Box-IO benchmark retains the box feasible region but introduces a multiplicative interaction between context and latent preference:

$$\max_{x\in[-1,1]^p} (\theta \circ u)^\top x, \tag{20}$$

where $\circ$ denotes element-wise multiplication. The optimal decision is again coordinate-wise,

$$x_j^\star = \mathrm{sign}(\theta_j u_j). \tag{21}$$

This benchmark is intended to test whether the inverse procedure can recover context-dependent objective effects that cannot be represented as a simple additive offset. Since the forward decisions depend primarily on the sign of the products $\theta_j u_j$, the magnitude of θ is generally less identifiable than its sign pattern. Consequently, normalization of the parameter vector is important when comparing estimated parameters.

Quadratic-Box-IO. The Quadratic-Box-IO benchmark introduces a strictly concave regularization term:

$$\max_{x\in[0,1]^p} (\theta + u)^\top x - x^\top x. \tag{22}$$

The unconstrained stationary point is

$$x = \frac{\theta + u}{2}, \tag{23}$$

and the box-constrained solution is obtained by coordinate-wise projection:

$$x^\star = \mathrm{clip}\left(\frac{\theta + u}{2}, 0, 1\right). \tag{24}$$

Unlike the linear benchmarks, the quadratic term typically produces an interior and unique optimizer. This results in substantially richer information about the magnitude of $\theta^\star$, making the problem more favorable for direct parameter identification.

Ball-IO. The Ball-IO benchmark considers a coupled Euclidean constraint:

$$\max_x (\theta + u)^\top x \qquad \text{s.t.} \qquad \|x\|_2 \le 1. \tag{25}$$

For $\theta + u \neq 0$, the optimizer is

$$x^\star = \frac{\theta + u}{\|\theta + u\|_2}. \tag{26}$$

Hence, all coordinates of the decision are coupled through the Euclidean norm.

This benchmark complements the coordinate-separable box problems by testing inverse recovery under a smooth, coupled feasible geometry. Because the direction of $\theta + u$ directly determines the optimizer, the benchmark provides a relatively informative setting for recovering the latent linear coefficient vector.

Overall, the five convex benchmarks contain no structural search over objective components or constraints. Their hypothesis space therefore contains only one structural model, and they primarily serve as controlled tests of parameter fitting, inverse optimality conditions, and consistency between the new implementation and the corresponding legacy solvers.

Remaining Results

Table 4: Results on the remaining Convex benchmarks. The evaluation metric is test regret; lower is better. C denotes clean observations and N denotes noisy observations.

Type	Dataset	KKA	MOM	ERM	ECHO
Hadamard Box	50-C	0.0000	0.0000	0.0000	0.0000
	50-N	0.2181	1.7514	0.0729	**0.0000**
	100-C	0.0000	0.0000	0.0000	0.0000
	100-N	2.1905	**0.0000**	**0.0000**	**0.0000**
Quadratic Box	50-C	0.0000	0.0000	0.0000	0.0000
	50-N	0.0033	0.4392	0.0023	**0.0015**
	100-C	0.0000	0.0000	0.0000	0.0000
	100-N	0.0022	0.4525	**0.0013**	**0.0013**
Ball	50-C	0.0000	0.0000	0.0000	0.0000
	50-N	0.0961	0.1511	0.0248	**0.0238**
	100-C	0.0000	0.0000	0.0000	0.0000
	100-N	0.0920	0.1119	**0.0112**	**0.0112**

Remaining Results Table 4 reports the results on Hadamard Box, Quadratic Box, and Ball. These benchmarks are comparatively easy from an inverse-identification perspective, and therefore provide limited discrimination among the considered methods. In particular, for Quadratic Box and Ball, the inverse feasible set collapses to a single point under the observed decisions, effectively determining the underlying parameter and causing different methods to converge to nearly identical, often zero-regret, solutions. Hadamard Box is similarly simple in a different sense: the observations identify primarily the sign pattern of the parameter, while the decision rule is invariant to positive rescaling, so recovering the relevant signs is sufficient to attain optimal decisions in many settings. Consequently, most methods achieve zero or near-zero regret on these datasets, and the remaining differences are substantially less informative than those observed on Simplex and Additive Box. To keep the main content focused on the more discriminative benchmarks and to save space, we report these remaining results in the appendix.

A.2 Combinatorial Inverse Optimization

The combinatorial benchmarks move from continuous decision spaces to discrete optimization. In these tasks, expert demonstrations are binary selections, matchings, or job sequences. The benchmarks are constructed to test not only parameter recovery but also objective-component selection and, in the most challenging cases,

Data protocol. For each benchmark and random seed, a single certified pool of instances is generated, and datasets with $N \in \{50, 100, 200\}$ observations are cut from it in a nested manner: the training and validation sets each contain $N/2$ instances, and the 100 clean test instances are identical across all N. To model imperfect experts, 15% of the training decisions are replaced by feasible but suboptimal decisions, assigned in rotation to three severity tiers with a relative optimality gap in $[1, 5]\%$, $[5, 15]\%$, and $[15, 30]\%$; validation and test decisions are left clean.recovery of hidden constraint structure.

A.2.1 Knapsack-IO

The Knapsack-IO benchmark considers $n = 20$ items. Each item j has an integer weight w_j and belongs to a group g_j. The binary decision $x_j \in \{0, 1\}$ indicates whether the item is selected. The feasible set is defined by the capacity constraint

$$\sum_{j=1}^{n} w_j x_j \leq B, \tag{27}$$

where

$$B = \max \left\{ \text{round} \left(\alpha \sum_j w_j \right), \max_j w_j \right\}, \qquad \alpha \in \{0.25, 0.5, 0.75\}. \tag{28}$$

The ground-truth benchmark uses a group-dependent reward model,

$$c_j^\star = \log(1 + w_j)\mu_{g_j}, \tag{29}$$

where the latent group multipliers μ_g are sampled once and shared across all problem instances. The expert solves

$$\max_{x\in\{0,1\}^n} \sum_{j=1}^{n} \log(1 + w_j)\mu_{g_j} x_j \qquad \text{s.t.} \qquad \sum_j w_j x_j \le B. \tag{30}$$

The inverse learner does not receive the group multipliers. Instead, it is given a library of candidate objective components. The principal ground-truth component is `group_multiplier`, which represents the group-specific functions

$$\sum_{j:g_j=g} \log(1 + w_j)x_j. \tag{31}$$

The library additionally contains several distractors, including the total number of selected items and quadratic interactions between items from the same or different groups.

A nominal component `log_weight_reward` is also included, but its feature vector lies in the column space of the group-specific multiplier features. It therefore acts as a redundancy test for the feature-reduction procedure rather than as a genuinely independent explanatory component.

The feasible-set structure is assumed known in this benchmark. Thus, the inverse problem focuses on recovering the objective structure and its parameters. The forward and inverse candidate models are solved using a MILP solver, with pairwise binary interactions linearized through McCormick constraints when quadratic terms are present.

The benchmark is designed to avoid a degenerate regime in which item weight alone determines the solution. If all rewards were a deterministic monotone function of weight, a simple baseline ranking could reproduce the expert decisions without recovering the latent reward structure. Introducing group-specific multipliers makes the preferred item ordering dependent on information unavailable to such a baseline.

A.2.2 Sparse Combinatorial Matching

We consider a sparse bipartite matching problem with $m = 20$ nodes on each side. Every left node a and right node b carries a group label $g_a^L, g_b^R \in \{0, 1, 2\}$, drawn uniformly at random. Candidate edges are sampled independently with an instance-specific density $p_E \in \{0.3, 0.5, 0.7\}$, so the graph is generally not complete. The decision is a binary vector $x \in \{0, 1\}^E$ selecting a set of non-conflicting edges, subject to the degree constraints

$$\sum_{e\in\delta(v)} x_e \le 1, \qquad \forall v \in V_L \cup V_R. \tag{32}$$

Unlike perfect matching, not every node is required to be matched. The degree constraints are known to the learner and are included in every candidate model.

For an edge $e = (a, b)$, define the observable group distance

$$d_e = |g_a^L - g_b^R| \in \{0, 1, 2\}. \tag{33}$$

The ground-truth objective combines a distance-dependent edge reward with a pairwise penalty on chosen edges whose left endpoints share a group:

$$\max_x \sum_{e\in E} r^\star_{d_e} x_e \;-\; \gamma^\star \sum_{\substack{e<e' \\ g^L_{a(e)}=g^L_{a(e')}}} x_e x_{e'} \qquad \text{s.t. degree constraints.} \tag{34}$$

The reward table is drawn once for the whole dataset from non-overlapping ranges $r_0 \in [8, 10]$, $r_1 \in [4, 6]$, $r_2 \in [1, 2]$, giving $r^\star = (9.71,\ 5.91,\ 1.54)$, and $\gamma^\star = 1$. The objective library contains three components: the ground-truth `reward_by_distance` (one weight per distance) and `same_group_pairs` (one non-negative weight), and a distractor `reward_by_right_group`, which rewards an edge according to the group label of its right endpoint only.

Because any two edges of a matching are vertex-disjoint, the pairwise term equals $\sum_g \binom{n_g}{2}$, where n_g is the number of chosen edges whose left endpoint lies in group g. The forward problem is therefore solved exactly as a MILP in which each n_g is encoded by one-hot count indicators, requiring $O(m)$ auxiliary variables instead of one linearization variable per edge pair; the pairwise form is already intractable at $m = 20$. The expert decisions are generated by the same solver and are certified optimal.

With three objective components and three candidate behavior models (exact, noisy, and suboptimal decisions), the search space contains $(2^3 - 1) \times 3 = 21$ structural hypotheses. The difficulty of this benchmark lies less in the size of the space than in identifiability. The reward table is identified only up to a positive affine map, and since the rewards take only three distinct values, ties are pervasive: between 85% and 100% of the generated instances admit more than one optimal matching. Two exact solvers can therefore return different matchings with identical objective values, so Hamming decision error can be large even for a model with zero regret. We consequently rank methods by regret and by the fraction of test instances solved to the true optimum, not by decision error.

A.2.3 SCHEDULING-IO

The Scheduling-IO benchmark considers single-machine scheduling with release times, heterogeneous job classes, and an unknown precedence topology. Each instance contains $n = 10$ jobs. Job j belongs to a group $g_j \in \{0, \dots, 4\}$ drawn uniformly at random, has processing time $p_j = \max\{1, \mathrm{round}(\mathcal{N}(\mu_{g_j}, 40^2))\}$ with group means $\mu = (100, 300, 500, 700, 900)$, and has release time $r_j \sim \mathcal{U}\{0, \dots, \lfloor \tau \sum_k p_k \rfloor\}$ with $\tau = 0.25$.

For a given job order, the corresponding non-delay schedule is constructed recursively. If job j is processed immediately after job k, its start time is

$$S_j = \max\{r_j, C_k\}, \tag{35}$$

and its completion time is

$$C_j = S_j + p_j. \tag{36}$$

The ground-truth objective is weighted total completion time:

$$\min \sum_{j=1}^{n} w_{g_j} C_j, \tag{37}$$

or equivalently, in the maximization convention used by the general framework,

$$\max \; -\sum_{j=1}^{n} w_{g_j} C_j. \tag{38}$$

The group-dependent weights $\{w^\star_g\}_{g=0}^4$ are drawn once from $\mathcal{U}[1, 5]$, giving $w^\star = (1.84, 2.41, 4.04, 2.53, 4.44)$, and are latent: they must be inferred from the expert schedules. The objective library contains the ground-truth `group_weights` component together with two distractors: unweighted total completion time $\sum_j C_j$ and a processing-time-weighted completion time $\sum_j (p_j/100)\, C_j$. Note that $\sum_j C_j$ lies in the span of the group-weight representation (it corresponds to equal weights), so it cannot be separated from `group_weights` by data; the library check flags it as redundant.

In addition to the objective, the benchmark contains a latent precedence structure defined on job groups. The learner is given three mutually exclusive candidate topologies, or none:

$$\mathcal{Q}_A : 0 \prec 1 \prec 2 \prec 3 \prec 4, \tag{39}$$

$$\mathcal{Q}_B : \{0, 1\} \prec 2 \prec \{3, 4\}, \tag{40}$$

$$\mathcal{Q}_C : \{0, 1\} \prec 2 \prec 4, \quad \{0, 1\} \prec 3 \prec 4, \tag{41}$$

where $\mathcal{Q}_B$ is the ground truth and $\mathcal{Q}_A, \mathcal{Q}_C$ are distractors. Precedence is enforced on the transitive closure: if group g precedes group h, then every job j with $g_j = g$ must finish before any job k with $g_k = h$ starts,

$$S_k \geq C_j. \tag{42}$$

Hence, the inverse task is twofold: the method must identify the objective representation and its group weights, while also recovering which precedence topology generated the demonstrations. With seven non-empty objective-component subsets, four precedence choices $G \in \{\varnothing, \mathcal{Q}_A, \mathcal{Q}_B, \mathcal{Q}_C\}$, and three candidate behavior models, the benchmark contains $7 \times 4 \times 3 = 84$ structural hypotheses (28 without the behavior dimension).

For $n = 10$, exhaustive enumeration of all $10! \approx 3.6 \times 10^6$ permutations is too slow to run inside the search loop. Since all completion times carry non-positive coefficients in the maximization form, the forward problem is instead solved exactly by a dynamic program over job subsets. A state is the set of already sequenced jobs; each state keeps the Pareto front of labels (completion time of the last job, accumulated objective), because a label that finishes earlier with no worse value dominates. Precedence is enforced through a predecessor mask per job. The dynamic program returns a certified optimum, and the expert schedules are generated with the same solver. Enumeration is used for $n \leq 8$, and a disjunctive big-M MILP is available for instances beyond the reach of the dynamic program.

The scheduling benchmark is deliberately constructed so that the hidden structure matters. Under unweighted completion time, or under topologies A and C, the earliest-completion-time dispatching rule already reaches a gap below 0.8%, leaving little information for inverse learning. On the weighted objective with topology B, the same rule has a gap of about 4.5% at $n = 10$, whereas it drops to about 0.3% when given the true weights. The combination of group-dependent completion-time weights and the selected precedence topology therefore creates a measurable behavioral gap between models that recover the latent structure and models that do not.

A.3 Routing

Dataset and task. The real-world experiment uses the 2021 Amazon Last Mile Routing Research Challenge dataset (Merchán et al., 2022). It contains 6,112 historical training routes driven in Seattle, Los Angeles, Austin, Chicago, and Boston, and 3,072 held-out evaluation routes. Each route starts and ends at a depot and visits a prescribed set of delivery stops, so it can be viewed as a restricted traveling-salesperson problem. The data include realized travel times, stop and package attributes, and an anonymized geographical Zone ID for each stop.

Zone-level inverse model. Following the routing formulation of Scroccaro et al. (2024), we first map each demonstrated stop sequence to a sequence of zones and treat each zone as a hypernode. This representation is appropriate because consecutive stops frequently share a Zone ID and the same zones recur across historical routes. A hypothesis specifies which available travel-time and historical zone-transition cost components enter the restricted-TSP objective. VeriHy fits their coefficients on the training routes, verifies candidate models on held-out routes, and uses route-level counterexamples to revise or combine the objective components. After predicting a zone order, the within-zone stop sequences are reconstructed to obtain a complete route.

Evaluation. For an observed stop sequence A and a predicted sequence B, the official challenge score is

$$\mathrm{Score}(A, B) = \mathrm{SD}(A, B) \frac{\mathrm{ERP}_{\mathrm{norm}}(A, B)}{\mathrm{ERP}_e(A, B)}, \tag{43}$$

where SD is sequence deviation, $\mathrm{ERP}_{\mathrm{norm}}$ is edit distance with real penalty using normalized travel times, and ERP_e is the corresponding number of edits. Lower values indicate closer agreement with the historical driver route. We report the mean score over held-out routes. Because the driver's true objective is unavailable, this benchmark measures behavioral reproduction rather than exact structural recovery.

A.4 Summary of Benchmark Roles

The benchmark suite spans several levels of inverse-optimization difficulty. The convex problems contain a fixed structure and primarily test parameter recovery and numerical correctness. Knapsack introduces a richer combinatorial objective library containing both redundant and nonlinear distractor components.

The scheduling benchmark provides the principal tests of structural discovery. The former jointly searches over objective features, feasibility constraints, and behavioral assumptions, whereas the latter requires simultaneous recovery of a weighted objective and a discrete precedence topology. Finally, matching benchmark illustrates a failure mode of decision-based inverse learning when the forward problem contains many optimal solutions.

Taken together, these benchmarks allow us to separately evaluate

$$\text{parameter recovery}, \quad \text{objective-structure recovery}, \quad \text{constraint-structure recovery}, \tag{44}$$

as well as the ability of the proposed method to distinguish genuinely explanatory structure from redundant or deliberately misleading candidate components.

B Problem Specification and Hypothesis Admission

Specification and compilation. The task specification declares the context and decision schemas, the base objective f_0, the mandatory constraints G_{base}, objective library $\mathcal{L}_{\mathrm{obj}}$, optional constraint library $\mathcal{L}_{\mathrm{con}} \setminus G_{\mathrm{base}}$, behavioral choices $\mathcal{R}$, decision loss Δ, and forward solver. A compiler maps each objective block to $\psi_k(x, u)$ and each constraint to a feasibility test and solver representation. It also declares parameter domains and gauge conditions Θ_Φ, which prevent arbitrary rescaling where required. The search code uses these compiled operators rather than task-specific branches. The hypothesis space comprises $\Phi \subseteq \mathcal{L}_{\mathrm{obj}}$, $G_{\mathrm{base}} \subseteq G \subseteq \mathcal{L}_{\mathrm{con}}$, and $R \in \mathcal{R}$. Structure and parameters are related by the nested objective

$$H^\star \in \arg\min_{H \in \mathcal{H}} J_{\mathrm{val}}\Big(H, \widehat{\theta}(H; \mathcal{D}_{\mathrm{tr}})\Big). \tag{45}$$

We reserve $\mathcal{D}_{\mathrm{val}}$ for search, a separate audit split for a one-time check of a fixed shortlist, and the test split for final reporting. The audit result does not modify $H^\star$ or any search setting.

Fixed comparison probes. Before search, construct a fixed set $\mathcal{P}$ of pairs (i, z): select training contexts and obtain alternative decisions z by solving the full candidate objective library with sampled parameter vectors under base constraints. The feature difference and per-coordinate scale are

$$d_{ik,g}(z) = \psi_{k,g}(y_i, u_i) - \psi_{k,g}(z, u_i), \tag{46}$$

$$s_{k,g} = \max\{\mathrm{Std}_{(i,z)\in\mathcal{P}}[d_{ik,g}(z)], \epsilon_s\}. \tag{47}$$

The same probes support component linting, scaling, rank tests, initialization, and pre-fit equivalence throughout a run. The probe set is fixed independently of validation feedback. A nonzero scale floor prevents division by zero; components with negligible variation can be flagged as uninformative on these probes, without claiming global unidentifiability.

Admissibility and empirical identifiability. Admit a proposal only if $\Phi \neq \varnothing$, mandatory constraints are retained, mutual-exclusion rules hold, the compiled solver and chosen fitter support the model, and the feasible set is nonempty on training instances. When the fitting formulation treats y_i as a feasible forward decision, require $y_i \in \mathcal{X}_G(u_i)$ for all training instances; a violated optional constraint is then rejected before fitting. Apply the same compile-and-check procedure to an M3 component before adding it to the candidate library. For the selected objective blocks, stack probe differences into

$$\Delta_\Phi(\mathcal{P}) = \big[d_{ik,g}(z)\big]_{(i,z)\in\mathcal{P},\, k\in\Phi,\, g}. \tag{48}$$

After accounting for the declared gauge and fixed base objective, reduce an empirically rank-deficient representation to an identifiable subset only when the reduced model preserves the same permitted objective family. Otherwise record the ambiguity and reject or retain the proposal according to a prespecified rule. A probe-rank test is an empirical screen, not a guarantee of global identifiability.

LLM interfaces. M0 takes the highest-ranked admissible evidence-based edit; M1 chooses a schema-constrained ID from an admissible menu; M2 returns one or two such edits in sequence; and M3 proposes a component in the task's specification language. The LLM input contains the specification, selected parent summaries, score components, counterexamples, and candidate edits,

but no ground-truth structure. The validator returns a reason for invalid output for a bounded number of repair attempts, after which the policy falls back to M0. The applicable level is a property of the benchmark interface: tasks with a closed, validated library support M0–M2, whereas M3 additionally requires executable checks for new components. Report the level actually used for each benchmark rather than assuming that all levels apply to all tasks.

C Parameter-Fitting Objectives

Write I_{tr} for training indices. For a comparison decision z, define

$$\Delta f_{0,i}(z) = f_0(y_i, u_i) - f_0(z, u_i), \tag{49}$$
$$\Delta \psi_{i,\Phi}(z) = \psi_\Phi(y_i, u_i) - \psi_\Phi(z, u_i), \tag{50}$$

where ψ_Φ concatenates selected feature blocks. Fitting uses only training demonstrations; the validation contexts and predictions are not part of these objectives.

For compactness, write $m_i(z;\theta) = \Delta f_{0,i}(z) + \theta^\top \Delta \psi_{i,\Phi}(z)$ for the objective margin between an observed decision and an alternative.

C.1 Exact Discrete Decisions

When R assumes an exactly optimizing expert, estimate θ by

$$\max_{\theta,\rho,\xi} \quad \rho - C \sum_{i \in I_{\mathrm{tr}}} \xi_i - \lambda_\theta \|\theta\|_1 \tag{51}$$
$$\text{s.t.} \quad m_i(z;\theta) \geq \rho - \xi_i, \quad i \in I_{\mathrm{tr}}, \quad z \in \mathcal{X}_G(u_i) \setminus \{y_i\}, \quad \xi_i \geq 0, \quad 0 \leq \rho \leq \rho_{\max}, \quad \theta \in \Theta_\Phi. \tag{52}$$

The parameter domain supplies bounds, signs, or a gauge as appropriate. Rather than enumerate alternative decisions, solve a restricted program and use the separation oracle

$$z_i^{\mathrm{sep}} \in \operatorname*{arg\,max}_{z \in \mathcal{X}_G(u_i) \setminus \{y_i\}} f_\Phi(z, u_i; \theta) \tag{53}$$

to add a violated margin constraint. Stop when none is found within the solver tolerance or the fitting limit is reached. If the feasible set has no alternative to y_i, that instance contributes no separation constraint. Solver truncation is recorded as such, rather than counted as an exact optimum certificate.

C.2 Noisy Discrete Decisions

For potentially suboptimal demonstrations, use a task-defined $\Delta(y_i, z) \in [0, 1]$ and solve

$$\min_{\theta,\xi} \quad \|\theta\|_1 + \frac{C}{|I_{\mathrm{tr}}|} \sum_{i \in I_{\mathrm{tr}}} \xi_i \tag{54}$$
$$\text{s.t.} \quad \Delta f_{0,i}(z) + \theta^\top \Delta \psi_{i,\Phi}(z) \geq \Delta(y_i, z) - \xi_i, \quad i \in I_{\mathrm{tr}}, \quad z \in \mathcal{X}_G(u_i), \quad \xi_i \geq 0, \quad \theta \in \Theta_\Phi. \tag{55}$$

The cutting-plane separation oracle maximizes the loss-augmented score

$$z_i^{\mathrm{LA}} \in \operatorname*{arg\,max}_{z \in \mathcal{X}_G(u_i)} \big[f_\Phi(z, u_i; \theta) + \Delta(y_i, z) \big]. \tag{56}$$

This fitter is available only when the compiled decision loss admits a compatible loss-augmented solver. Its margin constraint is defined relative to feasible demonstrations, even if those demonstrations are suboptimal.

C.3 Continuous Decisions

For differentiable continuous candidates with suitable constraint qualifications, write all selected constraints as $g_j(x,u) \le 0$. The stationarity residual for maximizing f_Φ is

$$r_i^{\mathrm{stat}} = \nabla_x[-f_\Phi(y_i,u_i;\theta)] + \sum_j \lambda_{ij}\nabla_x g_j(y_i,u_i). \tag{57}$$

We minimize

$$\mathcal{L}_{\mathrm{KKT}} = \sum_{i\in I_{\mathrm{tr}}} \|r_i^{\mathrm{stat}}\|_2^2 + \beta \sum_{i,j} \big(\lambda_{ij} g_j(y_i,u_i)\big)^2 + \gamma\|\theta\|_2^2 \tag{58}$$

subject to $\lambda_{ij} \ge 0$ and $\theta \in \Theta_\Phi$, with training demonstrations required to be feasible. KKT conditions are necessary under the relevant regularity assumptions and characterize global optima only for an appropriate convex feasible set and concave objective in this maximization convention. A nonconvex candidate would require an explicitly declared local-optimum interpretation; it is not certified by this residual.

D Verification and Counterexample Calculations

D.1 Held-Out Scoring

For each $i \in I_{\mathrm{val}}$, freeze $\widehat{\theta}_H$ and solve

$$\widehat{x}_i^H \in \operatorname*{arg\,max}_{x\in\mathcal{X}_G(u_i)} f_\Phi(x,u_i;\widehat{\theta}_H). \tag{59}$$

Use the task's decision loss to define

$$L_{\mathrm{dec}}(H) = \frac{1}{|I_{\mathrm{val}}|}\sum_{i\in I_{\mathrm{val}}} \Delta(\widehat{x}_i^H, y_i). \tag{60}$$

The treatment of solver ties and equivalent expert decisions must be fixed in Δ before search. If $\mathcal{X}_G(u_i)$ is empty, record an infeasibility event and use the task's declared maximal decision loss for that context instead of an undefined prediction.

For $I_H^{\mathrm{feas}} = \{i \in I_{\mathrm{val}} : y_i \in \mathcal{X}_G(u_i),\ \mathcal{X}_G(u_i) \ne \varnothing\}$, let

$$r_i(H) = \Big[f_\Phi(\widehat{x}_i^H,u_i;\widehat{\theta}_H) - f_\Phi(y_i,u_i;\widehat{\theta}_H)\Big]_+, \tag{61}$$

$$b_i(H) = |f_\Phi(\widehat{x}_i^H,u_i;\widehat{\theta}_H)| + |f_\Phi(y_i,u_i;\widehat{\theta}_H)|. \tag{62}$$

The revealed objective gap is

$$L_{\mathrm{gap}}(H) = \frac{\sum_{i\in I_H^{\mathrm{feas}}} r_i(H)}{\epsilon + \sum_{i\in I_H^{\mathrm{feas}}} b_i(H)}. \tag{63}$$

Define $L_{\mathrm{gap}} = 0$ if I_H^{feas} is empty; feasibility is penalized separately. This score captures a model's preference for a different solution, while decision loss also distinguishes choices with similar objective values.

Feasibility mismatch is

$$L_{\mathrm{feas}}(H) = \frac{1}{|I_{\mathrm{val}}|}\sum_{i\in I_{\mathrm{val}}} \mathbf{1}\{y_i \notin \mathcal{X}_G(u_i)\} + \frac{1}{|I_{\mathrm{val}}|}\sum_{i\in I_{\mathrm{val}}} \mathbf{1}\{\mathcal{X}_G(u_i) = \varnothing\}. \tag{64}$$

An empty candidate feasible set can incur both terms by design; the score weights must be calibrated with this convention in mind.

For F fixed training folds, refit the same H to obtain $\widehat{\theta}_H^{(1)},\dots,\widehat{\theta}_H^{(F)}$. With

$$d_{\cos}(a,b) = 1 - \frac{\langle a,b\rangle}{\|a\|_2\|b\|_2 + \epsilon}, \tag{65}$$

the instability term is

$$U(H) = \frac{2}{F(F-1)} \sum_{1 \le r < s \le F} d_{\cos}(\widehat{\theta}_H^{(r)}, \widehat{\theta}_H^{(s)}). \tag{66}$$

Use a fixed convention for zero-norm parameter vectors (e.g., zero distance if both are zero); all folds use the same parameter gauge. The normalized structure cost is

$$C(H) = \frac{|\Phi| + |G \setminus G_{\text{base}}|}{|\mathcal{L}_{\text{obj}}| + |\mathcal{L}_{\text{con}} \setminus G_{\text{base}}|}. \tag{67}$$

Set the score weights and normalizations on development data, then freeze them for the reported comparisons. The ground-truth objective or structure does not enter any verification term.

D.2 Structured Evidence for Revision

Let $\mathcal{C}_H = \{i \in I_{\text{val}} : \Delta(\widehat{x}_i^H, y_i) > \tau_\Delta\}$, with a prespecified numerical tolerance $\tau_\Delta \ge 0$. If it is empty, set residual-based signals to zero and use other admissible edits. For an unselected objective block $k \notin \Phi$, define

$$\bar{d}_{k,g}(H) = \frac{1}{|\mathcal{C}_H|} \sum_{i \in \mathcal{C}_H} \left[\psi_{k,g}(y_i, u_i) - \psi_{k,g}(\widehat{x}_i^H, u_i)\right], \tag{68}$$

$$\pi_k(H) = \max_g \frac{|\bar{d}_{k,g}(H)|}{s_{k,g}}. \tag{69}$$

Taking the signed average before its absolute value lets conflicting residuals cancel. π_k is a heuristic signal for an omitted term; it is not, without further assumptions, an exact reduced cost for the fitting objective. For a selected block $k \in \Phi$, define its normalized coefficient strength

$$a_k(H) = \max_g |\widehat{\theta}_{k,g}| s_{k,g}. \tag{70}$$

For each candidate constraint c, let $\mathbb{I}_c(x, u)$ test whether x satisfies it. Define

$$v_c(H) = \frac{1}{|I_{\text{val}}|} \sum_{i \in I_{\text{val}}} [1 - \mathbb{I}_c(y_i, u_i)], \tag{71}$$

$$e_c(H) = \frac{1}{|\mathcal{C}_H|} \sum_{i \in \mathcal{C}_H} \mathbb{I}_c(y_i, u_i)[1 - \mathbb{I}_c(\widehat{x}_i^H, u_i)]. \tag{72}$$

When $\mathcal{C}_H$ is empty, define $e_c(H) = 0$. A constraint with large v_c contradicts observed decisions; an inactive constraint with large e_c could rule out a wrong prediction without removing its corresponding demonstration. These statistics prioritize edits but cannot establish their benefit until the edited model is refitted and independently checked.

E Graph Construction and Search Details

E.1 Initialization, Edits, and Parent Selection

Seed the DAG with admissible singleton-objective hypotheses ranked by the fixed probe comparisons and, when admissible, a full-library hypothesis. No validation decision is used to hand-pick a benchmark-specific initial structure. For a node $H = (\Phi, G, R)$, the local action space is

$$\begin{aligned} &\textsc{AddObj}(k),\ \textsc{RemoveObj}(k),\\ &\textsc{AddCon}(c),\ \textsc{RemoveCon}(c),\ \textsc{SwapCon}(c, c'),\\ &\textsc{SetBehavior}(R'). \end{aligned} \tag{73}$$

Add-objective edits prioritize large π_k; remove-objective edits prioritize small a_k; contradicted active constraints prioritize removal; inactive constraints with high e_c and low v_c prioritize addition. When structural changes offer little evidence yet errors remain, the policy can consider a more permissive supported behavioral assumption. Each proposed action is subjected to the same admission procedure, no matter which policy produced it.

Let $\widetilde{J}_v$ be the normalized validation score of node v. Sample an anchor from

$$p_t(v) = \frac{\exp(-\widetilde{J}_v/T)}{\sum_{w \in \mathcal{V}_t^{\exp}} \exp(-\widetilde{J}_w/T)}, \tag{74}$$

where $\mathcal{V}_t^{\exp}$ is the set of expandable representatives. Additional parents favor different predictions. Define

$$D_{\mathrm{beh}}(v, w) = \frac{1}{|I_{\mathrm{val}}|} \sum_{i \in I_{\mathrm{val}}} \Delta(\widehat{x}_i^v, \widehat{x}_i^w). \tag{75}$$

Given already selected parents $\mathcal{P}_t$, choose the next to maximize

$$q(v) + S_t \min_{p \in \mathcal{P}_t} D_{\mathrm{beh}}(v, p), \tag{76}$$

where q is decreasing in J and $S_t \in [0, S_{\max}]$ grows with the number of expansions since the last improvement. This changes the weight of complementary behavior without treating syntax differences as useful diversity. Hyperparameters and stagnation windows are fixed before testing.

E.2 Fusion and Information Exchange

When selected parents disagree on validation predictions, generate a full-union child containing their objective components and a conservative partial-import child adding only the highest-priority missing block to the anchor. For full union, discard optional constraints falsified by observed parent evidence; resolve mutually exclusive constraints using the parent with lower feasibility loss; retain the most permissive compatible parent behavior. These are proposal rules, not shortcuts to acceptance: recompile, admit, fit, and verify each child. Skip fusion for identical validation behavior. Record all contributing parents and component provenance as edges to the resulting node. At the next iteration, parent summaries expose score terms, counterexamples, evidence ranks, and edits already tried; no parameter estimate is inherited without refitting.

E.3 Equivalence, Budget, and Return Value

Merge identical tuples (Φ, G, R). Before fitting, use probe feasibility signatures and the span of $\Delta_\Phi(\mathcal{P})$ as a proxy for redundancy: identical behavior models and compatible parameter domains are required before feature-span equivalence can suppress a proposal. In particular, equal probe spans alone do not prove global model equivalence under different coefficient bounds or sign constraints. After fitting, if nodes have the same validation predictions and nearly equal scores under a prespecified tolerance, retain the structurally simpler representative while preserving the provenance edges. Mark a representative terminal when every admissible local edit has been evaluated, rejected, or identified as equivalent.

Given specification $\mathcal{S}$, policy M, and budget B, the complete procedure is:

1. Compile libraries; construct fixed probes and scales; check the initial seeds. Fit and verify all admitted, distinct seeds.
2. While the budget remains and an expandable representative exists, select parents according to (74) and optionally (76). Pass their evidence to M to propose local edits and eligible fusion candidates.
3. Admit, reduce, and deduplicate each proposal. For every accepted new node, fit on $\mathcal{D}_{\mathrm{tr}}$, verify on $\mathcal{D}_{\mathrm{val}}$, save J_H and E_H, and add provenance edges. Invalid proposals are logged without a fit.
4. Return the lowest-J node and its fitted parameters. Evaluate a fixed top-K validation shortlist on the audit split only once and do not revise the returned model on that basis.

Specify whether B counts admitted fits, solver calls, or total compute and report all three. A run also stops if no expandable node remains or a prespecified stagnation condition is met. If comparing LLM policies, log tokens, invalid proposals, and repair attempts alongside common fit and forward-solve budgets.

F PROMPT REPRESENTATION AND EXAMPLES

This appendix documents the prompts used by the language-model proposal policy in VERIHY. The LLM is responsible only for proposing the next hypothesis to evaluate. Numerical fitting, feasibility checking, and scoring are performed by deterministic optimization and verification procedures.

The prompts are instantiated dynamically from the current search state. Bracketed text such as *[selected parents]*, *[verifier evidence]*, or *[candidate hypotheses]* denotes values inserted by the framework rather than literal text shown unchanged across calls.

In both convex and combinatorial inverse optimization, the LLM does not directly evaluate hypotheses on the validation set. Instead, it receives structured evidence derived by the verifier and uses this evidence to propose the next search step.

F.1 CONVEX INVERSE OPTIMIZATION

In the convex setting, each search node represents an estimator

$$h : (\mathbf{U}, \mathbf{Y}) \mapsto \widehat{\boldsymbol{\theta}},$$

where $\mathbf{U}$ and $\mathbf{Y}$ denote the observed training contexts and decisions. The proposal policy chooses among a set of unevaluated estimator candidates, or may synthesize a new estimator program when the candidate library does not cover the desired principle.

F.1.1 SYSTEM MESSAGE

Convex system message

You are the proposal policy of ECHO-IO, an evidence-conditioned hypothesis search for inverse optimization.
Each node of the search is an **ESTIMATOR**: a rule that maps training observations to `theta_hat`.
A deterministic verifier scores every estimator on held-out validation decisions (J, smaller is better); you never see the validation data itself, only the evidence the verifier extracts from it.
Every step you are assigned **ONE** operator, chosen by the search from the evidence; follow it.
Think about the geometry of the problem: which parameter vectors are consistent with the training decisions, and which of them is most likely to generalize.

F.1.2 USER-MESSAGE TEMPLATE

For a selected search state, the user message is assembled from the forward optimization problem, the estimator toolbox, the selected parents, verifier evidence, behavioural equivalence classes, the search operator, and the set of unevaluated candidates.

Convex user-message template

PROBLEM
An expert repeatedly solves a known convex optimization problem in dimension $p = [p]$.
For each instance i, the expert sees a context vector $u_i \in \mathbb{R}^p$ and chooses the optimal decision y_i.
[forward-model description for the current geometry]
The parameter vector $\theta \in \mathbb{R}^p$ is fixed but unknown.
You receive *[number of training observations]* training pairs (u_i, y_i) and must estimate θ so that the decisions $x^\star(u; \widehat{\theta})$ reproduce the expert on new instances.
[noise description]
/PROBLEM

VERIFIER
[verification criterion defining J] Smaller is better.
The evidence of each estimator also reports 3-fold cross-validation on the training data, including `L_cv`, `instability_U`, and per-coordinate counterexamples.
/VERIFIER

TOOLS AND LIBRARY
[deterministic estimator toolbox]
/TOOLS AND LIBRARY

SELECTED PARENTS
[one JSON record per selected parent, containing the estimator, its score, fitted parameter, training fit, validation evidence, and cross-validation diagnostics]
/SELECTED PARENTS

BEHAVIOURALLY DISTINCT ESTIMATORS
[representatives of previously observed prediction-equivalence classes, together with the number of variants producing the same predictions]
/BEHAVIOURALLY DISTINCT ESTIMATORS

RECENT FAILURES
[optional recent validation or execution error]
/RECENT FAILURES

OPERATOR
Operator suggested by the evidence: *[refine / repair_fit / generalize / explore / fuse]*.
[operator-specific instruction]
/OPERATOR

CANDIDATES
```
{"candidate_id": "candidate_0", "family": ...,
 "config": ..., "note": ...}
{"candidate_id": "candidate_1", ...}
...
```
/CANDIDATES

TASK
Choose the **ONE** candidate you expect to have the lowest J, or choose `"new_code"` and write a new estimator program when you have a concrete idea none of the candidates covers.
Return
```
{"choice": <candidate_id or "new_code">,
 "code": <program or "">,
 "principle": <2--6 words>,
 "reason": <one or two sentences>}.
```
A program defines `def estimate_theta(U, Y)` returning a vector of shape $(p,)$, computes θ from (U, Y) only, may call the provided deterministic tools, and must finish within the execution budget.
Budget left: *[remaining search decisions]*.
/TASK

The instantiated problem description depends on the forward geometry. For example, the simplex case specifies

$$\mathcal{X} = \left\{ x \in \mathbb{R}^p : x \geq 0,\ \sum_j x_j \leq 1 \right\},$$

with objective

$$\max_{x \in \mathcal{X}} (\theta + u)^\top x.$$

The prompt also provides the corresponding closed-form forward solution. Analogous descriptions are supplied for the Hadamard box, quadratic box, additive box, and Euclidean ball geometries.

F.1.3 Estimator toolbox

The proposal policy may combine predefined deterministic estimation routines or implement a new estimator.

Convex estimator toolbox

`kka(U,Y,...)`: approximate-KKT residual minimization using bounded linear least squares.

`mom(U,Y,...)`: maximum optimality-margin estimation.

`forward(U,theta)`: evaluate the closed-form forward optimizer.

`decision_error(X,Y)`: compute normalized per-observation decision error.

`feasible_set(U,Y,...)`: construct the inverse-feasible set of parameter vectors making all training decisions optimal.

`center(U,Y,...)`: select a central point of the inverse-feasible set using either a Chebyshev-center or sample-mean construction.

`erm(U,Y,...)`: empirical risk minimization of the induced decision loss.

`sign_lsq(U,Y,...)`: bounded least-squares estimation for sign-based forward models.

A proposed program may call these routines on subsets or resamples of the training data, combine multiple estimates, post-process them, or implement a new estimator from scratch.

F.1.4 Evidence-conditioned operators

At each search step, one operator is selected from the verifier evidence and included in the prompt.

Convex operator instructions

REFINE
Refine the anchor parent: propose a configuration change or small code change that the evidence suggests will lower J.

REPAIR_FIT
The anchor does not reproduce all training decisions. Propose an estimator whose $\widehat{\theta}$ better reproduces the training decisions; under noiseless observations, an exactly consistent parameter vector exists.

GENERALIZE
The anchor reproduces the training decisions, yet validation decisions are wrong. Prefer an estimator that selects a different, better-justified point from the set of training-consistent parameters, or one that corrects coordinates highlighted by cross-validation counterexamples.

EXPLORE
The search is stagnating or recent proposals reproduce predictions already seen. Prefer an estimator based on a qualitatively different principle rather than another local parameter modification.

FUSE
Several parents are selected. Prefer an estimator that combines useful properties of the parents and differs from each individual parent.

F.1.5 Illustrative convex prompt

The following shortened example shows the structure of a convex proposal step.

Example: convex estimator proposal

PROBLEM
An expert repeatedly solves a convex optimization problem in dimension $p = 10$.
Feasible set: $x \geq 0$ and $\sum_j x_j \leq 1$.
Objective: maximize $(\theta + u)^\top x$.
You receive 50 training pairs and must estimate θ such that the resulting decisions reproduce the expert on new instances.

The observations are noisy.
/PROBLEM

SELECTED PARENTS

```
{"id":"n2",
 "principle":"feasible-set centre",
 "J":0.3118,
 "evidence":{...}}
{"id":"n0",
 "principle":"KKT residual minimization",
 "J":0.3251,
 "evidence":{...}}
```

/SELECTED PARENTS

OPERATOR
fuse
Several parents are selected. Prefer an estimator that combines what works in them and differs from each parent.
/OPERATOR

CANDIDATES

```
{"candidate_id":"candidate_0",
 "family":"fuse",
 "config":{"fitter":"ensemble",
           "combine":"median",
           "members":[...]}}
{"candidate_id":"candidate_1",
 "family":"refine_anchor",
 "config":{"fitter":"center",
           "method":"chebyshev"}}
```

/CANDIDATES

TASK
Choose the ONE candidate you expect to have the lowest J, or propose `"new_code"`.
/TASK

A representative response is

Example model response


```
{ "choice":"candidate_0",
"code":"",
"principle":"ensemble median",
"reason":"Combining the center and KKT estimators through a robust
aggregation rule may reduce sensitivity to noisy observations." }
```


F.2 Combinatorial Inverse Optimization

In the combinatorial setting, each hypothesis is a structural model

$$H = (\Phi, G, R),$$

where Φ is the set of objective components, G is the declared feasible-set structure, and R is the behavioural model.

The numerical parameters associated with H are fitted by a deterministic solver. The LLM reasons only about structural hypotheses and does not directly estimate their numerical parameters.

F.2.1 SYSTEM MESSAGE

Combinatorial system message

You are an inverse-optimization modelling assistant.
You work on the **STRUCTURE** of a decision model only: which objective components, which feasible-set components, and which behavioural model explain an expert's decisions.
You never estimate numerical parameters and never compute a score: a deterministic solver fits the parameters and a deterministic verifier scores every hypothesis.
Answer with JSON only, conforming to the provided schema.

F.3 M1: CANDIDATE-EDIT REASONING PROPOSAL

M1 performs constrained structural reasoning over a set of legal, unevaluated edits constructed by the search procedure. Each candidate corresponds to a complete resulting hypothesis, and the LLM selects the candidate it expects to achieve the lowest verifier score.

M1 user-message template

PROBLEM
[problem description]
[optional note describing imperfect or suboptimal training decisions]
/PROBLEM

OBJECTIVE COMPONENTS Φ
`- [name]: [description] (parameter dim [d], sign [nonneg/free])`
`- ...`
/OBJECTIVE COMPONENTS

FEASIBLE-SET COMPONENTS G
Always declared (base): *[base constraints]*.

`- [optional constraint]: [description]`
/FEASIBLE-SET COMPONENTS

BEHAVIOURAL MODELS R
`exact`: observed decisions are optimal under the declared model.

`noisy_decision`: deviations from exact optimality are represented through a slack or hinge-loss mechanism.

`suboptimal_decision`: most decisions are assumed optimal while a minority may be feasible but suboptimal and are set aside during fitting.
/BEHAVIOURAL MODELS

SCORE
The deterministic verifier computes

$$J = L_{\text{val}} + \lambda_{\text{gap}} L_{\text{gap}} + \lambda_{\text{feas}} L_{\text{feas}} + \lambda_{\text{unc}} U + \lambda_{\text{complexity}} C.$$

Here, L_{val} is validation decision error, L_{gap} is the revealed optimality gap, L_{feas} measures feasibility violations, U measures parameter instability across folds, and C penalizes structural complexity.
/SCORE

EVIDENCE FIELDS
`pricing[k]`: how strongly validation counterexamples point toward objective component k.

`coefficient_strength[k]`: the fitted magnitude of an active objective component relative to its variation.

`viol_obs[c]`: the fraction of observed decisions violating constraint c.

`explains[c]`: the fraction of counterexamples in which the expert respects c while the current model prediction violates it.

`train_fit`: diagnostics of the fit on the training decisions, including whether the solution collapsed, the fraction of optimality violations, and the fraction trimmed under the suboptimal-decision model.

/EVIDENCE FIELDS

SELECTED PARENTS

```
### parent #1 (anchor)
hypothesis:  {"objective":[...],
              "constraints":[...],
              "behavior":...}

verification:  {"J":..., "L_val":..., "L_gap":...,
                "L_feas":..., "U":..., "C":...}

evidence:  {"n_counterexamples":...,
            "pricing":{...},
            "coefficient_strength":{...},
            "viol_obs":{...},
            "explains":{...},
            "train_fit":{...}}
```

[additional selected parents]

/SELECTED PARENTS

ALREADY EVALUATED

```
{"objective":[...], "constraints":[...],
 "behavior":..., "J":...}
...
```

/ALREADY EVALUATED

CANDIDATE EDITS

Each candidate is legal and unevaluated.

```
{"candidate_id":"candidate_0",
 "edit_type":add_obj|remove_obj|add_con|remove_con|
 swap_con|set_behavior|fuse,
 "edit":[...],
 "result":{...},
 "evidence_score":...}

...
```

/CANDIDATE EDITS

TASK

Choose the **ONE** candidate you expect to have the lowest J.
Return
`{"candidate_id": ..., "reason": ...}`.

/TASK

F.3.1 ILLUSTRATIVE M1 PROMPT

Example: M1 on a knapsack hypothesis

PROBLEM

An expert repeatedly selects a subset of items under a weight budget. Each item has an integer weight and a group label in $\{0, 1, 2, 3\}$.
The expert is assumed to maximize a reward of the selected items, possibly including rewards for pairs of selected items, subject to the budget.

/PROBLEM

OBJECTIVE COMPONENTS Φ

```
- group_multiplier
- item_count
```

```
- log_weight_reward
- quadratic_cross_group
- quadratic_same_group
/OBJECTIVE COMPONENTS

FEASIBLE-SET COMPONENTS G
```
Always declared: `["capacity"]`.
```
/FEASIBLE-SET COMPONENTS

SELECTED PARENT
{"objective":["log_weight_reward"],
 "constraints":["capacity"],
 "behavior":"suboptimal_decision"}

verification:
{"J":0.29459,
 "L_val":0.27,
 "L_gap":0.02917,
 "L_feas":0.0,
 "U":0.0,
 "C":0.2}

pricing:
{"group_multiplier":0.7506,
 "item_count":0.1811,
 "quadratic_cross_group":0.2991,
 "quadratic_same_group":0.1124}
/SELECTED PARENT

CANDIDATE EDITS
candidate_0:  fuse item_count + log_weight_reward
candidate_1:  add quadratic_same_group
candidate_2:  add quadratic_cross_group
candidate_3:  set behavior to noisy_decision
candidate_4:  add group_multiplier
/CANDIDATE EDITS

TASK
```
Choose the ONE candidate you expect to have the lowest J.
```
/TASK
```

A representative response is

Example M1 response

```
{ "candidate_id":"candidate_2",
"reason":"Adding quadratic_cross_group is supported by the verifier
evidence and may explain counterexamples not captured by the current
objective." }
```

F.4 M2: Compositional Reasoning Proposal

M2 allows the proposal policy to compose a short sequence of structural changes rather than choosing a single pre-enumerated edit. The model receives the same problem specification, structural library, selected parents, verifier evidence, and search history as M1. The task block is replaced by a request to construct one or two legal edits starting from the anchor parent.

This proposal language allows the policy to reason about interactions between structural changes. For example, it may remove a weak objective component and add a more strongly supported one within a single proposal.

M2: compositional reasoning prompt

CONTEXT

[same problem specification, structural component library, behavioural models, verifier evidence, selected parents, and already evaluated hypotheses as in M1]

/CONTEXT

TASK

Write a script of **one or two edits** applied to the anchor parent.

Allowed operations:

```
add_obj(k),              remove_obj(k),
add_con(c),              remove_con(c),
swap_con(c_out,c_in),    set_behavior(r).
```

A base constraint cannot be removed. A constraint swap must respect the corresponding exclusive group. The resulting hypothesis must be legal and must not already have been evaluated.

Return

```
{"edits":[
  {"op":..., "target":[...]},
  ...
 ],
 "reason":...}.
```

/TASK

F.5 M3: Generative Reasoning Proposal

M3 extends the proposal space beyond the predefined structural library. When the existing components do not capture a recurring pattern in the verifier evidence, the LLM may synthesize one new objective or constraint component using a restricted expression grammar.

The generated component is added to the anchor hypothesis and subsequently passed through the same deterministic fitting and verification pipeline.

M3: generative reasoning prompt

CONTEXT

[same problem specification, current structural library, behavioural models, verifier evidence, selected parents, and already evaluated hypotheses as in M1]

/CONTEXT

TASK

Propose **ONE new component** that the current library lacks and that the evidence suggests the expert uses.
The proposed component is added to the anchor parent.

/TASK

EXPRESSION GRAMMAR

An expression consists of

```
sum(IDX in SET: body)
```

terms combined with +, −, and constants.
The expression body may multiply context data by at most two decision variables.
Supported operations include

```
+, −, *, /, neg, abs, norm2, sin, cos, log1p,
```

and `indicator(comparison)`.
A generated constraint must have one of the forms

```
expr <= expr,    expr >= expr,    expr == expr.
```

The symbol `g` indexes a parameter block of dimension `dim`.

/EXPRESSION GRAMMAR

INSTANCE INTERFACE

Decision variable: *[decision variable]* over index set *[index set]*.

Derived variables: *[derived-variable specification]*.

Context keys and shapes: *[context schema]*.

Index sets: *[index-set specification]*.
Pairs of an index set may be accessed through `pairs(SET)`, with fields `first` and `second`.

/INSTANCE INTERFACE

OUTPUT

Return

```
{
"component_kind":"objective" | "constraint",
"objective_component":  ...  or null,
"constraint_component":  ...  or null,
"reason":  ...
}.
```

/OUTPUT

G THE USE OF LARGE LANGUAGE MODELS

In this study, Large Language Models (LLMs) are used not only to polish the writing of the paper but also as an integral component of the proposed framework, where they operate as autonomous agents. The research ideas, methodological design, analysis, and final manuscript were conceived, developed, and validated by the authors.