OPTDIFF: FORMULATING OPTIMIZATION PROBLEMS WITH MASKED DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Turning a natural language description into a solver-ready optimization model demands faithful interpretation of entities, quantities, and relations, as well as generation fast enough for practical use. Existing approaches generate and revise such formulations with autoregressive language models, whose sequential decoding makes long formulations and iterative correction increasingly costly. We observe that an optimization formulation involves three tightly coupled kinds of decisions: variable semantics, relation structure, and numerical grounding. To make these decisions explicit, we design an intermediate representation (IR) that separates them while preserving their dependencies through shared semantic declarations. On this IR, masked diffusion can reconstruct multiple decisions in parallel and revise selected regions through remasking. Building on it, we present OptDiff, which fixes variable semantics in a shared node, reconstructs objective and constraint structures in parallel with confidence-guided denoising, binds numerical values in a separate stage, and repairs flagged statements through localized remasking. Across four optimization modeling benchmarks, OptDiff achieves competitive modeling accuracy with the lowest generation time among the compared methods. Compared with a matched autoregressive baseline, it improves objective-value accuracy by 5.3 percentage points on average and reduces generation time by roughly $2\times$, reaching nearly $6\times$ on complex formulations. These results indicate that optimization formulations become well suited to diffusion once their semantic, structural, and numerical decisions are made explicit.

1 INTRODUCTION

Mathematical optimization supports operational decisions in production planning, logistics, staffing, and resource allocation, where plans are rebuilt whenever conditions change. Producing a model means turning a natural language description into decision variables, an objective, and constraints whose coefficients come from the quantities it states, and this remains difficult because every component is written against the same set of symbols. An entity read wrongly once reappears in every relation that refers to it and invalidates the formulation, so errors propagate easily and are expensive to correct. Automating this process therefore requires keeping symbols consistent across components, getting the relations right, and responding fast enough to be used.

Most existing methods approach this with prompting and agent workflows, decomposing the modeling task across roles and iterating with a solver. These workflows improve reliability but leave the generation mechanism unchanged, with repeated rounds of sequential generation and correction increasing inference time (Xiao et al., 2024; AhmadiTeshnizi et al., 2024b). Another line supplies the missing domain knowledge through training, using instruction data, structured intermediate representations, and process supervision (Huang et al., 2025a; Jiang et al., 2025; Zhou et al., 2026). Both build on autoregressive models, so generation cost grows with the size of the formulation and correcting erroneous regions requires sequential token generation. Balancing the generation cost against the accuracy of the resulting model therefore remains a challenge.

We observe that the objective and constraints describe distinct relations over shared variables and source quantities. Therefore, we design an intermediate representation (IR) that separates shared declarations, relation structures, and numerical assignments, making the semantic, structural, and numerical decisions of a formulation explicit. Shared declarations give concurrent components

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

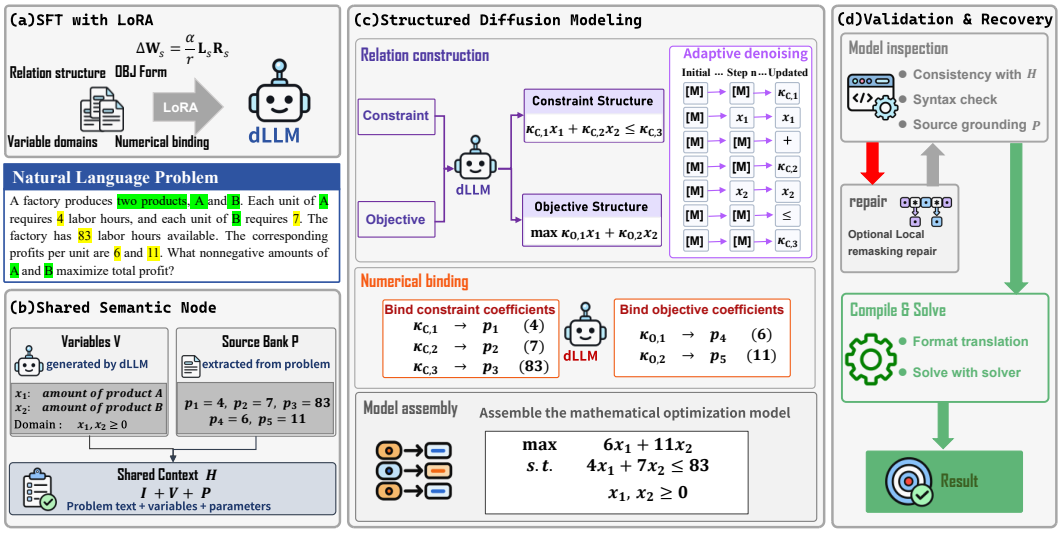

Figure 1: Overview of OptDiff. **(a)** Stage-specific LoRAs are trained on a frozen diffusion backbone for relation structure, source binding, objective form and domain labelling. **(b)** A shared semantic node pairs the generated variable records with a parameter bank extracted from the problem text, forming the shared context H. **(c)** Conditioned on H, constraint and objective structures are denoised in parallel over anonymous coefficient slots $\kappa_{f,\ell}$, which a separate stage binds to source parameters before assembly. **(d)** The assembled model is checked for consistency, syntax, and domains; flagged statements are repaired by localized remasking, and the result is compiled and solved.

consistent symbol meanings, while coefficient slots separate numerical choices from mathematical structure. Discrete diffusion language models can fill multiple masked positions in one update and reconstruct a masked region while keeping the surrounding content fixed (Nie et al., 2025; Ye et al., 2025). Once variable semantics are fixed, the remaining structural and numerical decisions are largely independent of one another, so on the IR these capabilities generate such decisions together and revise flagged statements in place. Through this representation, masked diffusion matches the decoupled structure of optimization modeling rather than serving merely as a faster decoder.

Specifically, OptDiff combines this IR with a masked diffusion large language model (dLLM) adapted through LoRA fine-tuning. The dLLM constructs a shared semantic node containing variable symbols, meanings, units, and initial domains, while a deterministic extractor builds a bank of source quantities from the problem description. Conditioned on this context, the model reconstructs objective and constraint structures concurrently. Adaptive denoising fills several positions selected by prediction confidence in each update and stops completed responses early, reducing the number of dependent generation updates. A separate diffusion stage then binds the coefficient slots to source quantities. After assembly, we check source evidence and predicted variable domains to identify statements that require correction. We remask those statements and preserve unaffected content during repair. A deterministic compiler translates the completed IR into a solver program. Our main contributions are as follows:

- We introduce OptDiff, a masked diffusion framework that exploits the structure of optimization models to construct solver-ready formulations substantially faster than autoregressive generation, with competitive accuracy.
- We design an intermediate representation that decouples the semantic, structural, and numerical decisions of an optimization model while preserving their dependencies, enabling masked diffusion to generate them stage by stage with intra-stage parallel decoding.
- We propose a localized repair mechanism that reconstructs only verification-flagged statements through masked diffusion, improving accuracy at little extra cost without altering verified content.

2 Related Work

Language models for optimization modeling. Work on automated formulation has moved from extracting linear programs from short descriptions toward producing executable models for problems with many variables, mixed integrality, and tabular data (Ramamonjison et al., 2022; Huang et al., 2025b; Yang et al., 2025b). Early systems treated the task as prompting. Chain-of-Experts assigns modeling subtasks to specialized roles with a conductor and backward reflection, while OptiMUS couples formulation with solver interaction and repair loops (Xiao et al., 2024; AhmadiTeshnizi et al., 2024b). A second line injects domain knowledge through training. ORLM fine-tunes open models on synthesized optimization instructions, while LLMOPT adds a five-element formalization with preference alignment and self-correction (Huang et al., 2025a; Jiang et al., 2025). Other approaches improve supervision and verification. StepORLM co-evolves a policy and a generative process reward model to evaluate complete modeling trajectories (Zhou et al., 2026). Opt-Verifier checks both modeling structure and solution validity (Liu et al., 2026a), while MIND uses localized errors to guide data synthesis and post-training (Liu et al., 2026b). These approaches improve prompting, model training, and verification, but they all retain autoregressive decoding. Autoregressive decoding generates the formulation one token at a time, so inference time grows with the length of the serialized sequence, whereas masked diffusion can reduce the number of sequential updates by filling multiple positions per denoising step. Accordingly, our method represents the semantic, structural, and numerical decisions of an optimization model separately in an IR and generates them in successive stages with this parallel decoding, so that the reduced number of updates translates into lower formulation time.

Repair strategy for autoformulation. Recent work combines language models with execution and verification feedback to repair optimization formulations. TriVAL verifies intermediate results throughout modeling and distinguishes local revision from reformulation (Fang et al., 2026). COVER uses solver counterexamples to guide targeted edits to a typed intermediate representation (Wu, 2026). PEARL learns when to invoke tools, revise the formulation, and terminate through interaction with execution environments and solvers (Lu et al., 2026). These approaches generate revisions autoregressively, producing replacement text one token at a time even when the edit is local. OptDiff performs local repair through masked reconstruction. It remasks selected fields or statements and reconstructs them in parallel, conditioned on the original problem and the retained formulation. Keeping the remaining content fixed prevents repair from disturbing correct statements, and parallel reconstruction keeps the cost of correction low.

Diffusion language models. Masked diffusion language models generate by iteratively unmasking positions under bidirectional context, and recent large-scale instances are competitive with autoregressive models of similar size (Nie et al., 2025; Ye et al., 2025; Nie et al., 2026). Three directions have shaped their decoding. Block Diffusion interpolates between autoregressive and diffusion generation by organizing output into blocks (Arriola et al., 2025); Fast-dLLM accelerates inference with caching and confidence-based parallel commitment (Wu et al., 2026); and parallel sampling with conditional-independence testing decides at runtime which positions may be committed together (Azangulov et al., 2026). Planned Diffusion combines an explicit plan with parallel chunk generation (Israel et al., 2026). These methods select parallel units through text blocks, token-level dependency tests, or learned plans for semantically independent chunks. Our method uses confidence-based adaptive denoising to commit many positions per update and thus accelerate decoding, and, once shared variable semantics are fixed, reconstructs the objective and constraints in separate concurrent branches to reduce interference between them.

3 Methodology

Given a natural language description, OptDiff constructs an optimization model through an intermediate representation that organizes shared declarations, mathematical relations, and numerical assignments. An overview of OptDiff is shown in Figure 1. An iLLaDA backbone with task supervision reconstructs these components (Section 3.2). Shared variable semantics provide the context for parallel relation generation and numerical binding (Sections 3.3 and 3.4). Local reconstruction updates selected fields before deterministic compilation and solver execution (Section 3.5).

3.1 Intermediate Representation for Masked Diffusion

An optimization formulation consists of mathematical relations over shared symbols. We design the intermediate representation (IR) to make these dependencies explicit for masked diffusion. Given the input description I, the formulation is $\mathcal{R} = (P, V, C, O)$, where P contains source parameters, V declares decision variables, $C = \{c_1, \ldots, c_m\}$ contains constraints, and O specifies the objective and its direction. A variable record $v_j = (x_j, \mu_j, u_j, d_j)$ associates symbol x_j with meaning μ_j, unit u_j, and domain d_j. A parameter record (p_k, ν_k, e_k) links identifier p_k to value ν_k and source clause e_k; in expressions, p_k denotes that value. For indexed formulations, an additional declaration table Γ records entity sets and admissible tuples referenced by parameters and decisions.

The shared context $H = (I, V, P)$ fixes the symbols and source quantities used by the constraint and objective branches, indexed by $\mathcal{F} = \{\mathrm{C}, \mathrm{O}\}$. Each branch $f \in \mathcal{F}$ constructs a relation skeleton S_f containing variables, operators, direction, source references, and coefficient slots $\kappa_{f,\ell}$, where ℓ indexes slots within the branch. Its binding B_f assigns a source parameter to each slot. This design organizes masked reconstruction around mathematical decisions rather than solver code. Combined with adaptive denoising, it allows multiple decisions to be completed in each update, reducing the sequential generation steps needed to construct the formulation. Shared declarations support concurrent relation construction, and fixed skeletons allow several numerical assignments to be predicted together. Statement boundaries and explicit fields define the regions remasked during local repair, with the remaining formulation providing visible context. The overall pipeline proceeds in five steps: (i) constructing the shared variable records and source bank, (ii) reconstructing the constraint and objective skeletons concurrently, (iii) binding their coefficient slots, (iv) performing local repair and assembly, and (v) compiling the formulation into a solver program:

$$I \xrightarrow{\text{(i)}} (V, P) \xrightarrow{\text{(ii)}} (S_{\mathrm{C}} \parallel S_{\mathrm{O}}) \xrightarrow{\text{(iii)}} (B_{\mathrm{C}} \parallel B_{\mathrm{O}}) \xrightarrow{\text{(iv)}} \mathcal{R} \xrightarrow{\text{(v)}} Y, \tag{1}$$

where $\parallel$ denotes concurrent reconstruction and Y is the program produced by deterministic compilation. Appendix A gives the record syntax, a complete scalar example, and the prompts.

3.2 Masked Diffusion with Task Supervision

Masked diffusion uses context on both sides of a missing token and can fill several positions in one update (Nie et al., 2025). We use iLLaDA 8B Instruct (Nie et al., 2026) with adapters trained for optimization modeling. For a generative stage $s \in \{\mathrm{var}, \mathrm{rel}, \mathrm{bind}\}$, let h_s be its prompt and $z_s^{(t)}$ the response after t denoising updates. The parameters θ_s specify the backbone and stage adapter. We omit the branch index when it is clear from context. At a masked position i, the model proposes $\hat{z}_{s,i}^{(t)} = \arg\max_\xi p_{\theta_s,i}(\xi \mid h_s, z_s^{(t)})$, where ξ ranges over the allowed tokens. An update rule selects masked positions $\mathcal{U}_s^{(t)}$ to fill:

$$z_{s,i}^{(t+1)} = \begin{cases} \hat{z}_{s,i}^{(t)}, & i \in \mathcal{U}_s^{(t)}, \\ z_{s,i}^{(t)}, & i \notin \mathcal{U}_s^{(t)}. \end{cases} \tag{2}$$

Each branch follows this rule with its own prompt and response. Visible context stays fixed throughout generation.

Training data combine the official NL4Opt training set (Ramamonjison et al., 2022), MAMO instances disjoint from the evaluation sets (Huang et al., 2025b), and OR-Instruct-Data-3K (Huang et al., 2025a). We construct the IR for each instance and retain only those that pass manual inspection and whose solver results match the reference answers. We first perform supervised fine-tuning (SFT) on variable records, constraints, objectives, and numerical modeling expressions. Training uses random token masks, fully masked responses, and semantic row masks. This supervision trains the model to recover the symbols, meanings, units, and initial domains in V. We then use LoRA (Hu et al., 2022) to train adapters for relation construction, numerical binding, objective forms, and variable domains.

Let y be a training target, $\mathcal{M}$ a nonempty set of sampled masked positions, and $\mathrm{mask}_{\mathcal{M}}(y)$ the resulting partial response. For $s \in \{\mathrm{rel}, \mathrm{bind}\}$, the corresponding adapter minimizes

$$\mathcal{L}_s = -\mathbb{E}_{(h_s,y)\sim\mathcal{D}_s,\, \mathcal{M}\sim q_s(\cdot|y)} \left[\frac{\sum_{i\in\mathcal{M}} w_{s,i} \log p_{\theta_s,i}(y_i \mid h_s, \mathrm{mask}_{\mathcal{M}}(y))}{\sum_{i\in\mathcal{M}} w_{s,i}} \right], \tag{3}$$

where $\mathcal{D}_s$ is the stage training set, q_s samples masks over editable positions, and $w_{s,i} > 0$ is the loss weight. For binding, $p_{\theta_{\text{bind}},i}$ is normalized over the source choices defined below. Objective forms and variable domains use cross-entropy on their categorical labels.

3.3 Shared Variable Semantics Node

The variable adapter reconstructs V from I, recovering decision quantities, their meanings, units, and initial domains. We assign symbols $x_1, x_2, \ldots$ in order of first mention and use the same variable records in the objective and constraint branches. The source parameter bank P is extracted deterministically from numbered clauses of I. Each parameter retains its value and source location, and exact conversions provide additional numerical forms. Together, I, V, and P form the shared node H. During relation construction and binding, H fixes the variable meanings, symbol identities, and source quantities used by both branches.

3.4 Parallel Relations and Numerical Binding

Adaptive denoising. Shared variable and relation generation use confidence guided updates. Each response is divided into fixed windows, with the first window containing masks selected as the active window. At a masked position i, the confidence is $\rho_{s,i}^{(t)} = p_{\theta_s,i}(\hat{z}_{s,i}^{(t)} \mid h_s, z_s^{(t)})$, for $s \in \{\text{var}, \text{rel}\}$. We include positions in the active window with $\rho_{s,i}^{(t)} \geq \eta$ in $\mathcal{U}_s^{(t)}$ of Equation 2, where η is a fixed threshold. We also fill a minimum number of the most confident positions to advance the window. Generation moves to the next window when the current one is complete. A response finishes when its end token and all preceding positions are filled, then leaves the active computation. The objective uses one window covering its response.

Let L_{A} be the autoregressive output length for a comparable formulation, and let K_{D} be the number of dependent denoising rounds used by OptDiff. Let T_{A} and T_{D} denote their decoding times, with average times per decoding step τ_{A} and τ_{D}. Their decoding time ratio is

$$\frac{T_{\text{A}}}{T_{\text{D}}} \approx \frac{L_{\text{A}}}{K_{\text{D}}} \frac{\tau_{\text{A}}}{\tau_{\text{D}}}. \tag{4}$$

In OptDiff, confidence guided denoising commits multiple positions in each update, reducing K_{D}. An autoregressive decoder generates the variable declarations, objective, and constraints one token at a time, so its dependent steps increase with L_{A}. When the resulting $L_{\text{A}}/K_{\text{D}}$ exceeds the relative step cost $\tau_{\text{D}}/\tau_{\text{A}}$, the reduction in dependent steps yields faster decoding.

Formulation structure reconstruction. Conditioned on H, the relation adapter reconstructs S_{O} and S_{C} concurrently. The objective branch specifies what to optimize and in which direction, while the constraint branch represents feasibility requirements and their bounds. Separate prompts and response boundaries keep these roles distinct, reducing confusion between objectives and constraints and helping retain boundary conditions. Both branches use the same variable records and source parameters in H. Each skeleton specifies its variables, operators, direction, and source references, with numerical values represented by slots $\kappa_{f,\ell}$ for the subsequent binding stage.

Binding numerical values. The completed skeleton defines a binding template. Its mathematical structure stays fixed while the parameter choices are reconstructed. A bijection χ maps the parameter identifiers p_k in P to the source choice tokens in $\mathcal{A}_P$. For branch f, the binding prompt $h_{\text{bind},f}$ contains H, S_f, and the choice mapping χ^{-1}. Let $z_{\text{bind},f}^{(t)}$ be its partial response after t updates, and let $i_{f,\ell}$ be the response position of slot $\kappa_{f,\ell}$. Write $\lambda_{f,\ell}^{(t)}(\xi)$ for the binding adapter's logit for choice token ξ at this position. The candidate distribution and resulting binding are

$$\begin{aligned} p_{\theta_{\text{bind}},i_{f,\ell}}(\xi \mid h_{\text{bind},f}, z_{\text{bind},f}^{(t)}) &= \frac{\exp \lambda_{f,\ell}^{(t)}(\xi)}{\sum_{\xi' \in \mathcal{A}_P} \exp \lambda_{f,\ell}^{(t)}(\xi')}, \quad \xi \in \mathcal{A}_P, \\ B_f(\kappa_{f,\ell}) &= \chi^{-1}(\hat{\xi}_{f,\ell}). \end{aligned} \tag{5}$$

Here $\hat{\xi}_{f,\ell}$ is the highest probability token committed to the slot. At each update, the binding adapter prioritizes unfilled slots by this candidate probability and keeps committed choices fixed. Both

branches use the same binding adapter and choice mapping. Training varies the correspondence between choice tokens and parameter identifiers so that the adapter learns to use the supplied mapping. At inference, the objective and constraint branches bind their slots concurrently. Substituting the selected parameters completes their numerical expressions.

3.5 Local Repair and Solver Assembly

The intermediate representation makes errors traceable to fields and statements. Before binding, we parse relations and check their variable names and source references against V and I. After binding, we check the assignments of coefficient slots to parameters in P. During compilation, we compare parameter values and inequality directions with their cited clauses. These diagnostics select the content for local repair.

Let F denote the diagnostics and $\mathcal{E}(F)$ the token positions selected for revision, including dependent expressions affected by the diagnosed error. For a response z, with reserved positions for missing fields, the masked input and repaired response z' satisfy

$$z_i^{\text{repair}} = \begin{cases} \texttt{[MASK]}, & i \in \mathcal{E}(F), \\ z_i, & i \notin \mathcal{E}(F), \end{cases} \qquad z'_i = z_i \quad \text{for all } i \notin \mathcal{E}(F). \tag{6}$$

Here `[MASK]` denotes the mask token. The model fills the selected positions using the original description and retained context. For a missing objective source, it reconstructs the source reference while keeping the objective expression fixed. We repeat the checks on the revised formulation and use any remaining diagnostics to guide further local repair.

We assemble the parameter, variable, constraint, and objective records into $\mathcal{R}$. Source evidence supplies explicit bounds, totals, and proportion relations, while the domain adapter and source checks refine variable domains. A deterministic compiler maps $\mathcal{R}$ to the solver program Y, resolving parameter references and translating the objective and constraints into solver operations. We execute Y and return results that pass structural checks, solve successfully, and have a finite objective value.

4 Experiments

We first introduce the datasets and evaluation metrics used in our experiments, followed by the implementation details. We then compare OptDiff with existing optimization autoformulation methods in terms of modeling accuracy and generation efficiency. Finally, we conduct ablation studies to examine the contribution of each component.

4.1 Datasets

We evaluate OptDiff on four optimization modeling benchmarks using the cleaned and verified test sets adopted by StepORLM (Zhou et al., 2026). **NL4Opt** (Ramamonjison et al., 2022). We use 213 validated natural-language optimization problems for evaluating formulation correctness. **MAMO-EasyLP** (Huang et al., 2025b). The evaluation set contains 545 instances with relatively simple LP and MILP structures. **MAMO-ComplexLP** (Huang et al., 2025b). We use 111 verified instances with more complex formulation structures and stronger dependencies among variables and constraints. **NLP4LP** (AhmadiTeshnizi et al., 2024b). The evaluation set contains 178 verified LP and MILP problems covering a range of operations research scenarios.

4.2 Metrics

We adopt two metrics to evaluate modeling correctness and generation efficiency. **Objective-value Accuracy (Acc.).** A generated formulation is considered correct if it can be successfully compiled and solved, and its optimal objective value matches the reference within $|\hat{y} - y| \leq 10^{-4} + 10^{-6}|y|$. Invalid or unsolved formulations are counted as incorrect. **Mean Generation Time (Time).** We report the average time required to generate the final formulation for each problem, including all generation calls performed by the method.

4.3 Implementation Details

We evaluate OptDiff on a single NVIDIA RTX 6000D GPU with 84 GiB of visible GPU memory, an Intel Xeon Platinum 8470Q CPU with an allocation of 25 vCPUs, and 92 GiB of RAM. All generation times in Table 1 are measured on this machine; methods based on GPT-4o call the API from it. Our implementation uses PyTorch 2.8.0 with CUDA 12.8 and Transformers 4.49.0. OptDiff builds on iLLaDA-8B-Instruct with supervised fine-tuning through LoRA. Inference uses BF16 precision and processes one problem at a time, with objective and constraint branches generated in parallel. We use adaptive denoising with a confidence threshold of 0.9 and early stopping upon completion. The maximum denoising budgets are 96 steps for shared variable semantics, 72 steps for relation generation, and 8 steps for numerical binding.

4.4 Comparison to State-of-the-Art Methods

Table 1: Accuracy (%, ↑) and mean generation time (↓) on four optimization benchmarks. Bold indicates the best reported value in each column.

Method	**NL4Opt**		**MAMO EasyLP**		**MAMO ComplexLP**		**NLP4LP**	
	Acc.	Time (s)	Acc.	Time (s)	Acc.	Time (s)	Acc.	Time (s)
Zero-shot LLMs								
GPT-4o (closed)	61.0	12.23	70.3	15.01	57.7	25.70	73.6	11.12
Qwen3-32B	77.5	46.97	92.3	45.71	46.8	90.49	93.8	47.25
Fine-tuned LLMs								
ORLM-Llama3 (8B)	90.6	13.35	91.6	15.57	63.1	27.30	88.2	17.50
StepORLM (8B)	**96.7**	21.54	**97.6**	22.07	**77.5**	30.32	**97.2**	25.66
LLMOPT-Qwen2.5 (14B)	85.4	22.98	87.7	22.94	64.0	40.02	86.5	33.13
Prompting and Agentic Methods								
Chain-of-Experts (GPT-4o)	66.7	67.74	94.3	82.80	50.5	75.38	86.5	62.01
OptiMUS-v0.3 (GPT-4o)	76.1	61.96	78.0	82.60	46.8	89.61	88.8	71.86
CAFA (GPT-4o)	68.1	7.52	71.2	7.56	45.0	9.48	50.0	8.66
Controlled Comparison								
Qwen3.5-9B + SFT	89.2	5.51	81.5	6.59	59.5	53.71	91.0	5.42
OptDiff (8B)	93.9	**2.81**	87.7	**3.07**	67.6	**9.00**	93.3	**2.79**

We compare OptDiff with representative optimization autoformulation approaches, including zero-shot LLMs such as GPT-4o (OpenAI, 2024) and Qwen3-32B (Yang et al., 2025a), fine-tuned language models, and prompting and agentic methods including OptiMUS-v0.3 (AhmadiTeshnizi et al., 2024a). We also include Qwen3.5-9B (Qwen Team, 2026) + SFT as a controlled autoregressive baseline using the same training data, intermediate representation, repair prompt, and hardware as OptDiff. We report both objective-value accuracy and mean generation time to assess modeling quality and efficiency.

As shown in Table 1, OptDiff achieves a favorable balance between modeling accuracy and generation efficiency. It improves both accuracy and generation time over the controlled baseline on all four benchmarks. The speed advantage becomes more pronounced on complex problems. On MAMO-ComplexLP, for example, OptDiff reduces mean generation time from 53.71 s to 9.00 s while also improving accuracy. Longer formulations increase the sequential decoding cost of the autoregressive baseline, widening the time gap with OptDiff. Compared with other fine-tuned models, OptDiff also offers substantially lower latency. On NL4Opt, its accuracy is close to StepORLM while its generation time is less than one-seventh as long. Among prompting and agentic methods, Chain-of-Experts and OptiMUS reach high accuracy on some benchmarks through repeated generation, verification, and solver interaction, but at a substantial cost in time, whereas the single-prompt CAFA (Deng

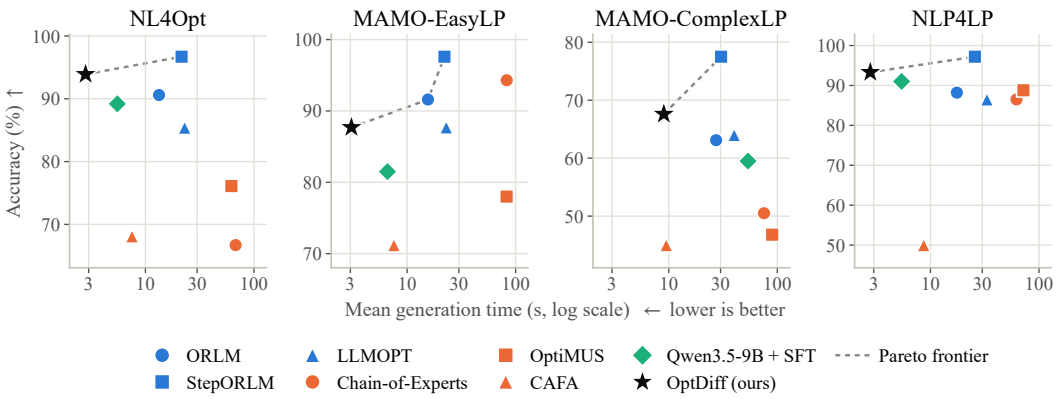

Figure 2: Accuracy versus mean generation time on the four benchmarks, using the results in Table 1 without the zero-shot LLMs. The time axis is logarithmic, and points toward the upper left are better. The dashed line connects the Pareto-optimal methods.

et al., 2024) is fast but considerably less accurate. Together, these results show that OptDiff reduces generation time across the benchmarks while maintaining strong modeling accuracy.

Figure 2 places each method by accuracy and generation time. OptDiff is the fastest method on all four benchmarks and lies on the Pareto frontier in every panel, outperforming both the controlled baseline and CAFA in accuracy and speed. The only methods with higher accuracy, StepORLM on all four benchmarks and ORLM and Chain-of-Experts on MAMO-EasyLP, require 3.4 to 27 times as much generation time.

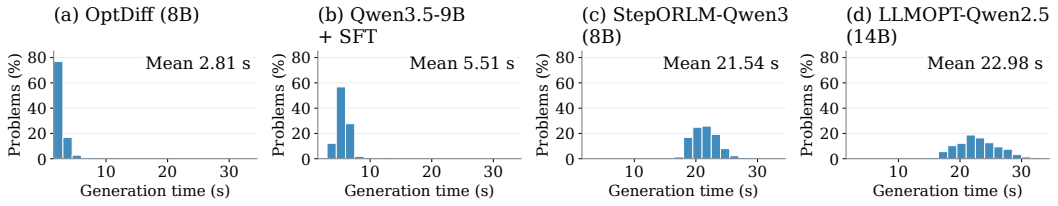

Figure 3: Generation time distributions on NL4Opt. **(a)** OptDiff (8B), **(b)** Qwen3.5-9B + SFT, **(c)** StepORLM-Qwen3 (8B), and **(d)** LLMOPT-Qwen2.5 (14B). Their mean times are 2.81, 5.51, 21.54, and 22.98 s, respectively.

Figure 3 shows how generation time varies across NL4Opt instances. Most OptDiff outputs fall in the earliest time bins, with only a sparse tail of slower cases. Qwen3.5-9B + SFT is concentrated in later bins, while StepORLM and LLMOPT are centered around 20 to 25 s. The shift across the distributions shows that OptDiff is faster for typical instances as well as on average.

4.5 Ablation Study

Table 2 adds the components of OptDiff cumulatively.

Representation and decoding. The structured IR gives the backbone a fixed record format. Applied to the raw model, it raises NL4Opt accuracy only from 7.5% to 14.1% but roughly halves generation time, suggesting that the compact format shortens responses. Task SFT then raises accuracy to 69.5% on NL4Opt and 60.6% on MAMO-EasyLP with little change in time, showing that modeling ability comes from supervision rather than the format alone. Adaptive denoising commits confident positions in parallel and stops completed responses. It makes generation 1.7 to 3.9 times faster without loss of accuracy, indicating that many IR positions can be committed together.

Structured modeling. The shared semantic node fixes the meaning of each symbol before relation reconstruction. Adding it raises accuracy from 69.0% to 80.3% on NL4Opt and from 61.1% to

Table 2: Cumulative ablations of OptDiff in accuracy (%) and mean generation time (s). Each row adds to the preceding configuration. Raw denotes iLLaDA 8B Instruct before task SFT, which is performed with LoRA.

Variant	NL4Opt		MAMO EasyLP		MAMO ComplexLP		NLP4LP	
	Acc.	Time (s)	Acc.	Time (s)	Acc.	Time (s)	Acc.	Time (s)
Raw (iLLaDA 8B)	7.5	29.49	2.8	31.29	0	38.56	9.0	29.95
+ Structured IR	14.1	15.73	9.5	16.34	0.9	19.07	14.0	14.63
+ Task SFT	69.5	15.55	60.6	14.61	49.5	20.07	73.6	15.88
+ Adaptive denoising	69.0	4.51	61.1	5.12	50.5	12.11	73.6	4.03
+ Shared semantics	80.3	4.02	73.4	4.73	58.6	11.33	79.2	3.89
+ Branch decoupling	85.9	4.08	78.0	4.12	62.2	11.51	82.0	3.92
+ Separate numerical binding	89.7	2.44	83.7	2.78	64.0	8.03	86.0	2.21
+ **Local repair (OptDiff)**	93.9	2.81	87.7	3.07	67.6	9.00	93.3	2.79

73.4% on MAMO-EasyLP, indicating that inconsistent symbol interpretation is a major source of error. Decoupling the objective and constraint branches raises accuracy further to 85.9% and 78.0% at similar cost, as separate prompts keep optimization goals apart from feasibility conditions. Separate numerical binding then selects coefficients from declared sources after the relations are fixed, raising accuracy to 89.7% and 83.7% while reducing time from 4.08 s to 2.44 s and from 4.12 s to 2.78 s. Selecting among source parameters is thus both more reliable and cheaper than generating values inside the relations.

Local repair. Local repair remasks only the fields flagged by verification and reconstructs them in parallel. It yields the largest gain on NLP4LP, from 86.0% to 93.3%, and raises NL4Opt accuracy from 89.7% to 93.9%, while adding less than 1 s per problem on every benchmark. Targeted correction therefore fixes diagnosed errors at small cost without altering verified statements.

5 Conclusion

We presented OptDiff, a masked diffusion framework for constructing optimization models from natural language. A shared semantic node anchors symbol meanings before objectives and constraints are reconstructed in parallel, followed by a separate stage for numerical binding. Adaptive denoising reduces generation time, while localized remasking repairs diagnosed statements without changing unaffected content. Across four benchmarks, OptDiff improves both modeling accuracy and generation time over a matched autoregressive baseline. Compared with the more accurate StepORLM, it offers substantially faster generation. These results indicate that optimization modeling becomes well suited to diffusion once an IR makes the semantic, structural, and numerical decisions of a formulation explicit. More broadly, making the decision structure of an output explicit may offer a general route to applying diffusion models to structured generation.

AI USE STATEMENT

In this work, we used generative AI tools for none of the tasks with required disclosure. We have not used generative AI tools for generating synthetic data sets, helping develop theoretical models or conceptual frameworks, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, implementing methods, assisting with translation, cleaning and reformatting data sets, or interpreting results, and formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, and supporting qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for editing the paper to improve readability and for searching for and identifying relevant literature. We have reviewed all AI-assisted work. The authors checked every AI-edited passage to ensure that its technical meaning was unchanged, and read and verified every paper identified with AI assistance against its original source before citing it. The language models evaluated in our experiments, such as GPT-4o and the Qwen models, are baselines or model backbones under study and did not assist in conducting this research. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

OptDiff is intended to assist the formulation of optimization models. Our experiments use existing public benchmarks and do not involve human participant studies. In applications such as staffing and resource allocation, objectives and constraints encode choices that can affect how benefits and burdens are distributed. Responsible use therefore includes domain expert review of the formulation and explicit consideration of the fairness of those choices. The intermediate representation makes variables, numerical sources, and mathematical relations available for this review.

REPRODUCIBILITY STATEMENT

Section 3 specifies the formulation pipeline, including the intermediate representation, adaptive denoising, numerical binding, and deterministic compilation. Section 3.2 describes the training data, training objectives, and LoRA adaptation. The verification rules and local repair procedure are given in Section 3.5, and Section 4.3 reports the hardware, software versions, and decoding budgets. Appendix A provides the record syntax, a complete scalar formulation example, and the prompt templates. All results in the tables are obtained from our runs on the same machine.

REFERENCES

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

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, 2024b. URL https://proceedings.mlr.press/v235/ahmaditeshnizi24a.html.

Marianne Arriola, Aaron Kerem Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block Diffusion: Interpolating between autoregressive and diffusion language models. In *The Thirteenth International Conference on Learning Representations*, pp. 50726–50753, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/7ede97c3e082c6df10a8d6103a2eebd2-Abstract-Conference.html.

Iskander Azangulov, Teodora Pandeva, Niranjani Prasad, Javier Zazo, and Sushrut Karmalkar. Parallel sampling from masked diffusion models via conditional independence testing. In *The Fourteenth International Conference on Learning Representations*, pp. 140485–140521, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/e389b15166cf98966ba058965a8c17e3-Abstract-Conference.html.

Haoxuan Deng, Bohao Zheng, Yirui Jiang, and Trung Hieu Tran. CAFA: Coding as auto-formulation can boost large language models in solving linear programming problem. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS '24*, 2024. URL https://openreview.net/forum?id=xC2xtBLmri.

Ziyang Fang, JinXi Wang, Jinghui Zhong, and Yew-Soon Ong. TriVAL: A tri-validation framework for faithful automatic optimization modeling. arXiv preprint arXiv:2605.23966, 2026. URL https://arxiv.org/abs/2605.23966.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. ORLM: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 73(6):2986–3009, 2025a. doi: 10.1287/opre.2024.1233. URL https://doi.org/10.1287/opre.2024.1233.

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

Daniel Israel, Tian Jin, Ellie Cheng, Guy Van den Broeck, Aditya Grover, Suvinay Subramanian, and Michael Carbin. Planned Diffusion. In *The Fourteenth International Conference on Learning Representations*, pp. 98298–98318, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/9fbae6362539ef7a151303c86cb51c00-Abstract-Conference.html.

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. LLMOPT: Learning to define and solve general optimization problems from scratch. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=9OMvtboTJg.

Haoyang Liu, Jie Wang, Boxuan Niu, Xiongwei Han, Yian Xu, Mingxuan Ye, Zijie Geng, Fangzhou Zhu, Tao Zhong, Mingxuan Yuan, and Jianye Hao. Opt-Verifier: Unleashing the power of LLMs for optimization modeling via dual-side verification. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://icml.cc/virtual/2026/poster/64663.

Weiting Liu, Han Wu, Yufei Kuang, Xiongwei Han, Tao Zhong, Jianfeng Feng, and Wenlian Lu. Automated optimization modeling via a localizable error-driven perspective. arXiv preprint arXiv:2602.11164, 2026b. URL https://arxiv.org/abs/2602.11164.

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

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *Advances in Neural Information Processing Systems*, volume 38, pp. 50608–50646. Curran Associates, Inc., 2025. doi: 10.52202/085713-1689. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/48b383b24230e0e6e649d9c98dae4d8c-Abstract-Conference.html.

Shen Nie, Qiyang Min, Shaoxuan Xu, Zihao Huang, Yuxuan Song, Yong Shan, Yankai Lin, Wayne Xin Zhao, Chongxuan Li, and Ji-Rong Wen. Improved large language diffusion models. arXiv preprint arXiv:2606.25331, 2026. URL https://arxiv.org/abs/2606.25331.

OpenAI. GPT-4o system card. arXiv preprint arXiv:2410.21276, 2024. URL https://arxiv.org/abs/2410.21276.

Qwen Team. Qwen3.5-9B. Hugging Face model card, 2026. URL https://huggingface.co/Qwen/Qwen3.5-9B. Accessed 2026-09-26.

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

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dLLM: Training-free acceleration of diffusion LLM by enabling KV cache and parallel decoding. In *The Fourteenth International Conference on Learning Representations*, pp. 57027–57051, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/5d8d4e6061c3ba96c240b7fa1ae3471d-Abstract-Conference.html.

Ziang Wu. COVER: Counterexample-guided repair for natural-language constraint modelling. In Hamido Fujita, Ali Selamat, and Farhan Mohamed (eds.), *New Trends in Intelligent Software Methodologies, Tools and Techniques*, volume 430 of *Frontiers in Artificial Intelligence and Applications*, pp. 172–182. IOS Press, 2026. doi: 10.3233/FAIA260737. URL https://doi.org/10.3233/FAIA260737.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan 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*, pp. 48519–48537, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/d45ee77826332c100a1e15f7765b99ff-Abstract-Conference.html.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. arXiv preprint arXiv:2505.09388, 2025a. URL https://arxiv.org/abs/2505.09388.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. OptiBench meets ReSocratic: Measure and improve LLMs for optimization modeling. In *The Thirteenth International Conference on Learning Representations*, pp. 24726–24759, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/3deb687c44d3687ace0729e5db3b4efd-Abstract-Conference.html.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7B: Diffusion large language models. arXiv preprint arXiv:2508.15487, 2025. URL https://arxiv.org/abs/2508.15487.

Chenyu Zhou, Tianyi Xu, Jianghao Lin, and Dongdong Ge. StepORLM: A self-evolving framework with generative process supervision for operations research language models. In *The Fourteenth International Conference on Learning Representations*, pp. 6914–6940, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0bcfb525c8f8f07ae10a93d0b2a40e00-Abstract-Conference.html.

A Intermediate Representation and Prompts

A.1 Intermediate Representation

The formulation is represented by $\mathcal{R} = (P, V, C, O)$, where P contains source parameters, V contains decision variable records, C contains constraints, and O specifies the objective. The problem description I, variable records V, and parameter bank P form the shared context $H = (I, V, P)$ for relation construction and numerical binding.

A variable record $v_j = (x_j, \mu_j, u_j, d_j)$ specifies a symbol, meaning, unit, and domain. Variable generation writes its semantic description as a `VAR` record; the assembled IR uses a `DEC` record for its solver declaration. Each parameter record (p_k, ν_k, e_k) links a value to its source clause. Parameter identifiers follow source clause order, then numerical value and transformation within each clause. Indexed formulations additionally use a declaration table Γ for entity sets and admissible tuples. The compiler translates the assembled records into a solver program Y.

For each branch $f \in \mathcal{F} = \{\mathrm{C}, \mathrm{O}\}$, the relation skeleton S_f contains variables, operators, and coefficient slots $\kappa_{f,\ell}$. Each standalone `?` in the generated skeleton denotes one slot. Slots are named `ck1`, `ck2`, and so on for constraints, and `ok1`, `ok2`, and so on for the objective, in their order of appearance. A coefficient multiplying a grouped expression remains one binding choice. Literal zero and structural coefficients of one can appear directly.

Numerical binding B_f assigns a parameter from P to each slot. Distinct slots may select the same parameter. A bijection χ assigns each available parameter a distinct choice code, and the binding prompt supplies the inverse mapping χ^{-1}. The response template fixes the slot identifiers and masks their choice positions. Substituting the selected parameters into S_f gives the completed relations.

A.2 A Complete Scalar Formulation

Consider the following source clauses.

```
C001: For producing chairs and tables, a chair uses 2 units of wood, a
    table uses 3 units of wood, and 18 units of wood are available.
C002: Choose nonnegative integer numbers of chairs and tables to
    maximize profit, earning 5 units per chair and 4 units per table.
```

The variable records and source parameters are

```
VAR x1 | number of chairs produced | chairs | nonnegative_integer
VAR x2 | number of tables produced | tables | nonnegative_integer

PARAM p1 = 2 @C001
PARAM p2 = 3 @C001
PARAM p3 = 18 @C001
PARAM p4 = 4 @C002
PARAM p5 = 5 @C002
```

The relation branches generate

```
CON : + ? x1 + ? x2 <= + ? @C001
OBJ maximize : + ? x1 + ? x2 @C002
```

With $\chi(p_1) = \mathtt{A}$, $\chi(p_2) = \mathtt{B}$, $\chi(p_3) = \mathtt{C}$, $\chi(p_4) = \mathtt{D}$, and $\chi(p_5) = \mathtt{E}$, the bindings are

```
BIND ck1 = A
BIND ck2 = B
BIND ck3 = C
BIND ok1 = E
BIND ok2 = D
```

Substitution gives the assembled IR.

```
PARAM p1 = 2 @C001
PARAM p2 = 3 @C001
PARAM p3 = 18 @C001
PARAM p4 = 4 @C002
PARAM p5 = 5 @C002
DEC x1 scalar nonnegative_integer
DEC x2 scalar nonnegative_integer
CON c1 : + p1 x1 + p2 x2 <= + p3 @C001
OBJ maximize : + p5 x1 + p4 x2 @C002
```

The corresponding model is

$$\max_{x_1, x_2 \in \mathbb{Z}_{\geq 0}} \quad 5x_1 + 4x_2 \qquad \text{subject to} \qquad 2x_1 + 3x_2 \leq 18. \tag{7}$$

The bindings assign p_5 to chair profit and p_4 to table profit. The optimum is $(x_1, x_2) = (9, 0)$ with objective value 45.

A.3 Prompt Templates

The following instruction texts are used for shared variable generation, relation construction, numerical binding, objective form prediction, domain prediction, and objective source repair. Fields enclosed in angle brackets are replaced with the content for the current problem. Each prompt is supplied as a user message via the backbone's chat template with an assistant generation prefix.

Shared variable definitions. The response provides the meanings, units, and initial domains used to construct V.

```
Read the optimization problem. Define ONLY the decision variables, in
    order of first mention. A variable is a quantity we choose, not a
    fixed parameter. Give its entity, unit, and domain. Do not solve the
    problem. Emit one line per variable, exactly in this format:
VAR x1 | number of units of product A produced | units | nonnegative_real
VAR x2 | number of units of product B produced | units | nonnegative_real
Use nonnegative_integer only when whole items or integrality are
    required.
PROBLEM:
<problem description>
VARIABLE DEFINITIONS:
```

Constraint relations. The response gives the constraint skeleton with anonymous slots.

```
Read the optimization problem and predict only its constraints relation
    structure. Use exactly the supplied decision variables with their
    shared meanings. Include a variable in a relation only if that
    entity participates in the stated relationship. Preserve signs,
    comparisons, objective direction, ratio denominators, and source
    clauses. Replace every source parameter occurrence by the standalone
    token ?. Every occurrence is independent; do not generate slot IDs.
    Do not choose parameter IDs or numeric coefficients yet. Literal
    zero and structural unit coefficients may remain. Return only lines
    in the form CON : expression <= expression @C001, with the
    appropriate comparison and evidence. Do not generate constraint IDs.
    Then stop.
QUESTION (numbered source clauses):
<numbered source clauses>
SHARED VARIABLE ENTITIES:
<shared variable meanings>
<variable declarations>
SOURCE PARAMETER BANK (available for interpreting the problem; do not
    copy p IDs into the skeleton):
<source parameter bank>
CONSTRAINTS RELATION SKELETON:
```

Objective relation. The objective branch receives the same shared information and generates its relation skeleton.

```
Read the optimization problem and predict only its objective relation
    structure. Use exactly the supplied decision variables with their
    shared meanings. Include a variable in a relation only if that
    entity participates in the stated relationship. Preserve signs,
    comparisons, objective direction, ratio denominators, and source
```

```
    clauses. Replace every source parameter occurrence by the standalone
    token ?. Every occurrence is independent; do not generate slot IDs.
    Do not choose parameter IDs or numeric coefficients yet. Literal
    zero and structural unit coefficients may remain. Return one OBJ
    maximize : expression @C001 or OBJ minimize : expression @C001 line
    with the appropriate evidence, then stop.
QUESTION (numbered source clauses):
<numbered source clauses>
SHARED VARIABLE ENTITIES:
<shared variable meanings>
<variable declarations>
SOURCE PARAMETER BANK (available for interpreting the problem; do not
    copy p IDs into the skeleton):
<source parameter bank>
OBJECTIVE RELATION SKELETON:
```

Numerical binding. Both branches use the same binding instruction. The codebook pairs each choice code with a parameter identifier.

```
Bind every role slot to the source parameter with the correct entity,
    resource, and unit. Keep the mathematical structure and select only
    a code from the codebook.
QUESTION:
<problem description>
SHARED VARIABLE ENTITIES:
<shared variable meanings>
<variable declarations>
SOURCE PARAMETER BANK:
<source parameter bank>
RELATION SKELETON:
<relation skeleton with named slots>
BINDING CODEBOOK:
<choice code = parameter identifier, one per line>
Choose one code for each fixed BIND slot. Preserve all anchors.
```

For the scalar example, the constraint binding response has the fixed template

```
BIND ck1 = [MASK]
BIND ck2 = [MASK]
BIND ck3 = [MASK]
```

Objective form. The adapter selects an unweighted sum of all variables or the objective generated by the relation branch.

```
Read the optimization problem and the shared decision-variable meanings.
    Determine the requested mathematical objective form. Distinguish the
    quantity to optimize from capacities, requirements, and resource
    limits. An unweighted sum uses coefficient one for every shared
    variable exactly once, no other terms and zero offset. A subset of
    variables is not the sum of all variables.
A = minimize the unweighted sum of all shared decision variables.
B = maximize the unweighted sum of all shared decision variables.
C = another objective, including a subset, non-unit weights, or an
    offset.
QUESTION:
<problem description>
SHARED VARIABLE MEANINGS:
<shared variable meanings>
<variable declarations>
Return only one code A, B, or C.
OBJECTIVE FORM:
```

Variable domains. The domain adapter predicts a categorical label for each variable from the problem and candidate equations.

```
Determine the domain of each decision variable from the original
    optimization problem and the candidate equations. The equations may
    have mistakes. Infer what quantity each variable represents. Do not
    solve the optimization problem.
Counts of indivisible objects require integers; divisible weights,
    durations and flows are continuous. A yes/no choice is binary.
    Respect explicit relaxations or integrality. If the variable meaning
    or domain is ambiguous, answer unknown. Number of machines differs
    from machine operating hours.
Use exactly one of the following codes for each variable:
A = continuous
B = integer
C = binary
D = unknown

PROBLEM:
<problem description>

MODEL (domains intentionally hidden):
<candidate equations with variable domains removed>

Complete the answer slots with the appropriate codes only.
```

For two variables, the response template is

```
x1: [MASK]
x2: [MASK]
```

Four cyclic code mappings give four predictions per variable. A unanimous integer prediction with minimum confidence 0.55 changes a continuous declaration to the corresponding integer domain.

Objective source repair. The objective expression and its `@` marker remain fixed; the repair stage completes the missing source reference.

```
Read the numbered problem clauses and the fixed objective
expression. Identify the clauses that support the objective
direction and expression.

Complete only the source reference after the existing @ marker.
Return clause identifiers such as C001 or C001,C003, then stop.
Keep the supplied objective expression unchanged.
```

The selected identifiers are checked against the numbered source clauses before the relation proceeds to binding.