EVOOPT-LLM: EVOLVING INDUSTRIAL OPTIMIZATION MODELS WITH LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Supply-chain production and transportation models must account for changing production, workforce, inventory, and delivery rules while retaining the relationships among existing decisions. We present EvoOpt-LLM, an LLM-based approach to constructing an initial optimization model, injecting additional business constraints, and predicting candidate zero fixings before numerical solution. Three independently trained LoRA adapters generate formulations and code from natural-language problems, updated models from LP files and business rules, and variable-fixing lists from the current LP text. The supply-chain application centers on linking new rules to existing production decisions and conditioning fixing predictions on the model being solved. We evaluate the modules separately. A multi-category construction corpus measures basic modeling ability: with 3,000 examples, generation reaches 91%, while executability and reference-result matching among generated models reach 65.9% and 26.37%. A workforce-update case shows how adaptation changes the organization of a generated business-rule extension. With 400 pruning examples, variable-label F1 peaks near 0.56 on models of around 350–400 lines and declines on longer inputs. The results characterize LLM assistance for initial formulation, business-rule updates, and candidate variable fixing in supply-chain modeling.

1 INTRODUCTION

Supply-chain production and transportation planning coordinates what to produce, where to procure materials, how much inventory to hold, and when to transport and deliver goods. These decisions are linked by demand requirements, bills of materials (BOMs), material balances, and shared resource capacities. Mixed-integer linear programming (MILP) expresses these relationships in a form that numerical solvers can process. Its use depends on constructing the formulation and maintaining it as production, workforce, inventory, and delivery rules change. A model that represented yesterday's operating conditions may require substantial revision before it can support today's plan.

Consider a new workforce restriction in an existing production model. Writing a labor-capacity inequality is only part of the modeling task. The restriction must also be connected to the production decisions already present: if an operating state consumes labor, production must depend on that state. Otherwise, a solver can satisfy the new inequality without changing the production plan. Updating a business model therefore requires interpreting the new rule in the context of its variables and constraints. The update may also change which decisions are used. Variables predicted to be inactive before the change need not remain suitable for fixing afterward.

Recent work on LLM-based optimization modeling translates natural-language descriptions into formulations or implementation code, using structured reasoning, solver interaction, or task-specific training (Ramamonjison et al., 2023; AhmadiTeshnizi et al., 2023; Xiao et al., 2024; Huang et al., 2025; Jiang et al., 2024; Lu et al., 2025). Other work modifies modeling programs to improve their computational properties (Yuan et al., 2026), or predicts partial variable assignments for subsequent numerical solution (Nair et al., 2021). These approaches address related but distinct needs. Our focus is the preparation of an optimization model for changing supply-chain requirements: obtaining an initial formulation, expressing additional business rules within an existing model, and proposing variable fixings for the current problem.

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.

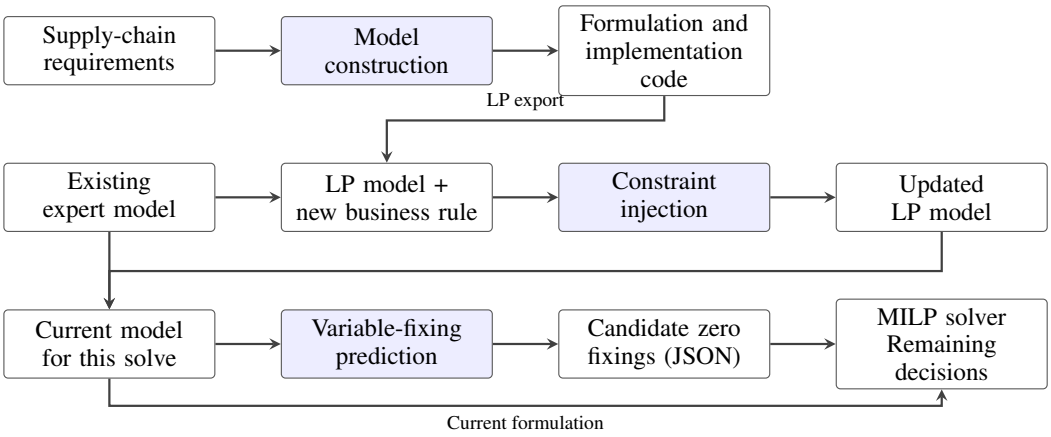

Figure 1: Supply-chain modeling workflow supported by EvoOpt-LLM. An existing expert model can enter at business-rule injection or variable-fixing prediction. The solver receives the current formulation together with candidate fixings. Arrows describe compatible task interfaces; experiments evaluate the modules separately.

We present EvoOpt-LLM for these tasks, using three independently trained LoRA adapters on a common language model. Construction returns a formulation and implementation code; injection returns an updated LP model; pruning returns candidate zero fixings for the current LP. A numerical solver determines the remaining decisions. Expert-built models can enter directly at injection or pruning. Here, *evolving* refers to modifying the optimization model during business use, not continuous language-model training or evolutionary search.

The contributions are threefold: we formulate these model-maintenance tasks and their interfaces; implement task-specific generation of formulations, business-rule extensions, and named fixing lists; and evaluate construction across training-set sizes, workforce-response excerpts, and pruning on compact supply-chain instances. The construction corpus covers general optimization problems, while updates and pruning provide the supply-chain application setting.

2 Related Work

Automated optimization modeling. Natural-language modeling combines problem interpretation, mathematical formulation, and program synthesis (Austin et al., 2021; Ramamonjison et al., 2023). Structured reasoning and repeated generation provide general tools for handling intermediate decisions (Wei et al., 2022; Wang et al., 2023; Yao et al., 2023). OptiMUS, Chain-of-Experts, and OptimAI develop optimization-specific modeling workflows (AhmadiTeshnizi et al., 2023; Xiao et al., 2024; Thind et al., 2025), while Mamo evaluates mathematical modeling with solvers (Huang et al., 2024). ORLM, LLMOPT, OptMATH, and LLaMOCO investigate specialized training and data generation for formulations or optimization code (Huang et al., 2025; Jiang et al., 2024; Lu et al., 2025; Ma et al., 2026). OR-R1 incorporates solver feedback into test-time reinforcement learning (Ding et al., 2025). Our construction experiment studies this basic capability across training-data sizes; it supplies an entry point for subsequent work on an existing model.

Modifying models and solution procedures. A modeling program can be edited to correct an error, to obtain a computationally stronger formulation, or to express a changed business requirement. These edits have different targets. FormuEvo uses solver-guided search over modeling programs to discover more efficient formulations (Yuan et al., 2026). FunSearch, ReEvo, Evolution of Heuristics, and LLaMEA concern program or heuristic design (Romera-Paredes et al., 2024; Ye et al., 2024; Liu et al., 2024; van Stein & Bäck, 2025). In the injection task studied here, the input model is instead extended to represent an additional operational rule. Success requires relating that rule to existing decisions and respecting the intended scope of the update.

Learning-assisted variable fixing. Learning can use recurring instance structure to guide optimization (Bertsimas & Stellato, 2021; Li et al., 2024). Neural Diving predicts partial assignments and uses a MIP solver for the remaining variables (Nair et al., 2021). Constraint Matters combines problem-level and instance-level information to select constraints for tightening (Li et al., 2026). These works motivate predicting decisions about the search space before numerical solution. Our pruning module learns a mapping from LP text to variable-name lists for zero fixing. Its evaluated target is agreement with reference annotations, with the mathematical consequences of jointly applying the fixings treated separately.

3 Supply-chain setting and task formulation

The application concerns production and transportation plans across items, plants, and periods. Production consumes components specified by a BOM and competes for shared resources. Procurement and transportation supply materials, inventory links availability across periods, and delivery satisfies demand. Substitution permits selected alternatives in the production or delivery structure. These relationships explain why an apparently local update can affect decisions elsewhere: restricting production through workforce availability can change material consumption, inventory, and the timing of deliveries.

We use the existing supply-chain formulation as the context for business updates and pruning, and write its MILP representation as

$$\mathcal{M}: \quad \min_{x\in\mathcal{F}} c^\top x, \qquad \mathcal{F} = \{x \in \mathbb{R}^n : Ax \le b,\ \ell \le x \le u,\ x_j \in \mathbb{Z}\ (j \in \mathcal{I})\}. \tag{1}$$

The vector x collects the modeled decisions; A, b, and the bounds encode their relationships. LP files serialize the objective, rows, bounds, and variable domains, including binary and general integer domains. The notation summarizes the existing mathematical program rather than introducing an additional supply-chain model.

Obtaining an initial model. For a natural-language specification d, construction produces a formulation and implementation code, $(\widehat{F}, \widehat{K}) = G_c(d)$. The code can be executed to instantiate the model and, for a MILP, export an LP file. The construction data also cover optimization classes beyond the supply-chain application. An existing expert formulation supplies an alternative starting point.

Adding a business rule. For a model $\mathcal{M}$ and an additional restriction r, the injection target is

$$\mathcal{M}^+: \quad \min_{x,y} c^\top x \quad \text{s.t.} \quad x \in \mathcal{F}, \quad y \in \mathcal{Y}, \quad g_r(x, y) \le 0, \tag{2}$$

where y denotes auxiliary decisions and g_r is linear. The training target retains the original objective expression and constraint rows while adding the rule and any linking constraints. This implies $\mathrm{proj}_x(\mathcal{F}^+) \subseteq \mathcal{F}$. Retaining the objective expression does not retain the old optimal value. Requests that relax an existing rule, or change costs, lie outside this append-only formulation. In particular, allowing interrupted production or multiple deliveries by augmentation requires that the original model already permit those choices.

Proposing fixings for the current model. Let $\mathcal{M}_{\mathrm{cur}}$ be the business model being prepared for solution, with feasible set $\mathcal{F}_{\mathrm{cur}}$. For a predicted index set $\widehat{\mathcal{P}}$, the solver receives the restriction

$$\mathcal{F}_{\mathrm{cur},\widehat{\mathcal{P}}} = \{x \in \mathcal{F}_{\mathrm{cur}} : x_j = 0 \text{ for every } j \in \widehat{\mathcal{P}}\}. \tag{3}$$

This is a candidate restriction for the current solve. When the original problem has an optimum, its optimal objective is retained exactly when the joint fixings leave at least one original optimum feasible. Preserving the entire feasible set requires every selected variable to be zero in every feasible solution. Both conditions apply to the fixing set as a whole.

4 EvoOpt-LLM

EvoOpt-LLM implements the tasks in Section 3 with a common pretrained language model and three independently trained LoRA adapters. Table 1 summarizes their supervised mappings and the

Table 1: Inputs and supervised response targets of the three adapters. The final column specifies how each output enters the optimization workflow.

Task	Input prompt contents	Supervised response target	Subsequent use
Construction	Task instruction and natural-language problem	Mathematical formulation and implementation code	Process and execute code; export an instantiated MILP to LP
Constraint injection	Task instruction, existing LP model, and additional business rule	Complete augmented LP model	Use the updated model as the current formulation
Variable pruning	Task instruction and current LP model	JSON array of `"variable_name=0"` strings	Map identifiers to variables and impose candidate zero fixings

subsequent use of their outputs. We describe the shared training procedure before specifying each module's data and inference procedure.

4.1 Task-specific adapter training

Let $k \in \{c, i, p\}$ denote construction, constraint injection, and variable pruning. Each task has instruction–response pairs

$$\mathcal{D}_k = \{(p_n^{(k)}, a_n^{(k)})\}_{n=1}^{N_k}, \tag{4}$$

where $p_n^{(k)}$ contains the instruction and input, and $a_n^{(k)}$ is the reference response: a formulation and implementation, an augmented LP model, or a JSON fixing list. Appendix C describes the task datasets.

For a pretrained matrix $W_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, LoRA (Hu et al., 2022) gives

$$W_k = W_0 + B_k A_k, \qquad A_k \in \mathbb{R}^{r_k \times d_{\text{in}}}, \quad B_k \in \mathbb{R}^{d_{\text{out}} \times r_k}, \tag{5}$$

with $r_k \ll \min(d_{\text{in}}, d_{\text{out}})$ and fixed scaling absorbed into the factors. The pretrained parameters θ_0 remain frozen; only the low-rank factors ϕ_k are trained.

The training code masks prompt and padding labels. Let $\mathcal{T}_n$ contain the response-token positions retained after truncation, including an end-of-sequence token when retained. The conditional next-token loss for a batch $\mathcal{B}$ is

$$\mathcal{L}_k(\phi_k; \mathcal{B}) = -\frac{1}{T_{\mathcal{B}}} \sum_{n \in \mathcal{B}} \sum_{h \in \mathcal{T}_n} \log P_{\theta_0, \phi_k}(a_{n,h}^{(k)} \mid p_n^{(k)}, a_{n,<h}^{(k)}), \qquad T_{\mathcal{B}} = \sum_{n \in \mathcal{B}} |\mathcal{T}_n|. \tag{6}$$

The loss is normalized by the number of non-ignored target tokens. Appendix B details the training configuration and accumulation convention. Each adapter is trained independently on its task's reference responses.

At inference, the requested task selects the adapter. Figure 1 shows the compatible interfaces: constructed MILPs can be exported to LP, LP models can receive business-rule updates, and the current LP can be used to predict fixings. Expert-built models can enter directly at injection or pruning.

4.2 Automatic model construction

Training data. A construction example contains a natural-language description d, reference formulation F, and implementation code K:

$$(p^{(c)}, a^{(c)}) = (\text{construction instruction} + d,\ \text{text}(F, K)). \tag{7}$$

Here $+$ denotes text concatenation. The formulation specifies index sets, parameters, variables and domains, the objective, and constraints. The code implements these definitions through the solver API. Names, indices, domains, and expressions must correspond across the two representations. Table 2 gives the construction corpus categories.

Generation and response processing. The adapter generates the formulation and a COPT implementation in one response. Response processing extracts the Python code block and appends a reporting block that reads the solver status and objective value. The extracted model definitions are then executed with COPT, and the reported result is compared with the reference answer using the procedure in Appendix C.

Execution and model export. The code is executed to instantiate and solve the model. An instantiated MILP can also be exported to LP for injection or pruning. Section 5.2 evaluates complete generation, successful execution, and reference-result matching as distinct aspects of construction performance.

4.3 Business-constraint injection

Training data. An example contains an existing model $\mathcal{M}$, an additional business restriction r, and a reference augmented model $\mathcal{M}^+$. Writing $L(\mathcal{M})$ for LP serialization gives

$$(p^{(i)}, a^{(i)}) = (\text{injection instruction} + L(\mathcal{M}) + r,\ L(\mathcal{M}^+)). \tag{8}$$

The target is the complete updated LP: it retains the original objective, rows, bounds, and domains, and adds the requested variables and restrictions. Appendix A gives the mathematical templates; Appendix C describes the editing records and parameter variation.

Constructing the extension. The generated content must identify the affected existing decisions and their indices and domains, express the restriction using those decisions and any required auxiliaries, and connect the auxiliaries to the original variables. The added rows have the form

$$D_r x + E_r y \le h_r, \qquad y \in \mathcal{Y}, \tag{9}$$

within Equation 2. They may constrain existing variables x, auxiliary variables y, or both. Linking rows ensure that restrictions on auxiliary decisions affect the corresponding production or transportation quantities. The adapter generates these components together in a single response.

Generating the updated model. At inference, the existing LP and business rule prompt the adapter to generate a candidate complete LP. A solver-ready formulation requires numerical coefficients, the original model components, the added rows, and variable declarations. The LP text is extracted from the objective declaration through the closing marker, retaining bounds and variable domains.

The append-only scope in Section 3 preserves the objective expression and does not relax existing constraints. Cross-period production and split-delivery restrictions therefore require models that already permit those timing choices. Section 5.3 and Appendix D examine workforce-response excerpts.

4.4 Variable pruning through candidate zero fixing

Training data. A pruning example pairs the current LP text with a reference list Z^{ref} extracted from a precomputed optimal solution $x^\star$ of the associated optimization model:

$$(p^{(p)}, a^{(p)}) = (\text{pruning instruction} + L(\mathcal{M}_{\text{cur}}),\ \text{JSON}(Z^{\text{ref}})). \tag{10}$$

The reference list selects variables satisfying $|x_j^\star| \le \epsilon_0$, where ϵ_0 is the zero-value tolerance, and records their LP identifiers in a JSON array of `"variable_name=0"` strings. Equation 6 trains the adapter to predict this solution's zero pattern from objective terms, rows, bounds, and domains. Appendix C.1 describes the compact instances and reference-label construction.

Prediction and application. The adapter generates a candidate list $\widehat{Z}$. To apply it, match the identifiers to the current model and form

$$\widehat{\mathcal{P}} = \{j : \text{name}(x_j) \in \text{Names}(\widehat{Z})\}, \qquad x_j = 0 \quad (j \in \widehat{\mathcal{P}}), \tag{11}$$

where Names extracts identifiers from zero-assignment entries. Equalities or fixed bounds impose the restriction in Equation 3. When zero is in a variable's domain, substitution removes its terms from the objective and affected rows; the solver determines the remaining decisions.

Use after a model update. When pruning follows injection, its input is the updated LP, so predictions reflect the decisions required by the new rule. Section 3 defines the joint conditions for feasible and optimum-preserving fixings. Section 5.4 evaluates recovery of the reference fixing lists.

5 EXPERIMENTS

The experiments address three questions: whether task-specific adaptation improves initial-model construction, how it changes business-rule injection into an existing supply-chain model, and how well the pruning adapter recovers reference zero fixings across LP-file lengths. Table 3 summarizes the results of these task-level evaluations.

5.1 EXPERIMENTAL SETUP

All experiments use openPangu-Embedded-7B-V1.1 (Chen et al., 2025) with independently trained task-specific LoRA adapters. Training and inference run on four Huawei Ascend 910B2 NPUs, each with 64 GB of HBM. Transformers provides model and LP generation. COPT executes the generated construction programs and supplies the objective values compared with the dataset reference answers. Appendix B lists the environment, adapter-training configuration, and response-loss implementation.

The construction corpus contains 3,000 examples with the category distribution shown in Table 2. Its categories include both mathematical-programming classes and application families; a scheduling or network problem may also admit a MILP formulation. To examine the effect of training-set size, we train separate adapters on subsets of 300, 600, . . . , 3,000 examples. All variants and the untuned base model are evaluated on the same held-out test set.

The Constraint Injection dataset contains 3,000 examples: 600 for each of five rule families—setup time, workforce availability, batch and inventory limits, production across periods, and split deliveries. Each example pairs an existing LP and a business-rule request with a reference LP response. Section 5.3 compares workforce-response excerpts.

The pruning experiment uses 400 training examples from the 413-record LP-VarPrune collection. Its compact supply-chain instances retain selected demand items together with their associated material, substitution, and resource relationships. The task predicts the reference zero-fixing list from each LP input. We examine performance against LP-file line count as a measure of textual input size. Appendix C describes the task collections and compact-instance preparation.

Table 2: Problem categories in the 3,000-example construction corpus. This corpus measures general modeling ability and is distinct from the supply-chain editing and pruning data.

Category	Share	Category	Share
Linear programming	22%	Scheduling optimization	9%
Integer programming	14%	Stochastic/risk optimization	7%
Mixed-integer linear programming	33%	Nonlinear optimization	3%
Network optimization	12%	Total	100%

5.2 AUTOMATIC CONSTRUCTION ACROSS TRAINING-SET SIZES

The construction experiment examines whether task adaptation improves the production of complete implementations, their executability, and their agreement with reference results. Let N be the number of test inputs, N_g the number yielding complete generated implementations, N_e the number of generated implementations that execute, and N_a the number whose evaluated result matches the reference. We report

$$G = \frac{N_g}{N}, \qquad E = \frac{N_e}{N_g}, \qquad A = \frac{N_a}{N_g}. \tag{12}$$

Generation rate G is measured over all test inputs. Executability E and reference-result matching A are measured over generated implementations. When no implementation is generated, the plotted values of the latter two metrics are set to zero.

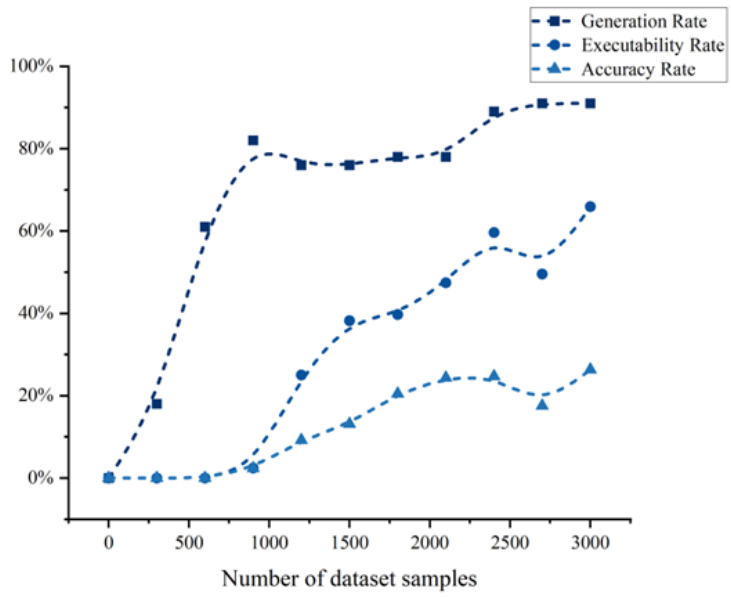

Figure 2: Automatic construction across training-set sizes. Generation is measured over all test inputs, while executability and reference-result matching are measured over generated implementations. The accuracy curve corresponds to reference-result matching.

Numerical matching compares the objective value returned with an optimal solver status against the reference objective. Both values are rounded to the nearest integer before applying a relative tolerance of 0.05, or an absolute tolerance of 0.05 when the rounded reference is zero. Multiple optimal decision vectors are evaluated through their objective values. Appendix C specifies the complete decision rule and treatment of execution outcomes.

Figure 2 shows an overall improvement with task adaptation, although the curves fluctuate across training-set sizes. The untuned base model scores zero on all three metrics under this evaluation. Generated implementations appear after training on 300 examples. Executable and reference-matching outputs become more frequent as the training set grows, with a marked increase in executability around 1,200–1,500 examples.

With 3,000 training examples, generation reaches 91%, while executability and reference-result matching among generated implementations reach 65.9% and 26.37%, respectively. These conditional rates correspond to approximately 60.0% executable outputs and 24.0% reference-matching outputs over all test inputs.

The difference between executability and reference-result matching is important for optimization modeling. A program may execute while using an incorrect capacity expression, variable domain, objective, or set of business restrictions. Reference-result matching evaluates agreement with the reference answer after execution. The results show that both executable and reference-matching outputs become more frequent with larger training sets.

5.3 Business-rule update: workforce constraint

The injection case examines a request to add workforce restrictions to an existing production and transportation model. The input contains demand, production-resource, production–transportation balance, distribution-center balance and capacity, and business-logic blocks. Figures 4 and 5 compare response excerpts before and after task-specific adaptation.

The untuned response represents labor use through an aggregate expression involving production and transportation quantities. It notes that the input lacks an explicit time dimension and formulates a global workforce restriction. The adapted response instead introduces resource requirements, a

Table 3: Results of the separate module evaluations. Construction rates follow Equation 12; constraint injection is illustrated through a qualitative workforce-update comparison; pruning F1 measures agreement with reference fixing annotations.

Module	Data setting	Observed result
Construction	3,000 training examples	$G = 91\%$, $E = 65.9\%$, $A = 26.37\%$
Constraint injection	3,000 records; five rule families	Qualitative workforce-update comparison: explicit resource and operating-state declarations
Variable pruning	400 training examples	Peak F1 ≈ 0.56 at about 350–400 LP lines

period set, period-specific workforce availability, and operating variables named `operation_A` and `operation_B`. It also retains the visible grouping of the original constraint blocks.

These declarations organize the workforce extension around resource requirements, temporal availability, and operating decisions. Appendix A.2 gives the corresponding workforce-capacity and production-linking template. The comparison indicates that task-specific adaptation improves the model's ability to express a new business rule in the structural context of an existing optimization formulation.

5.4 Variable-fixing prediction across model sizes

The pruning experiment compares the predicted zero-fixing set with the reference zero pattern extracted from the precomputed optimal solution for each instance. A true positive is a predicted variable that appears in this reference set, a false positive is a predicted variable outside the set, and a false negative is a reference variable omitted by the prediction. We report

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN}, \qquad F_1 = \frac{2TP}{2TP + FP + FN}. \tag{13}$$

Precision measures the fraction of proposed fixings contained in the reference list. Recall measures how much of that list is recovered, and F_1 combines these two aspects. For instances with multiple optimal solutions, the stored optimal solution defines the reference zero pattern used consistently across predictions.

Figure 3 shows that performance depends strongly on input length. After training on 400 examples, F_1 reaches approximately 0.56 for LP files of around 350–400 lines. The adapter recovers some reference fixings for inputs up to roughly 800 lines, but precision and recall generally decline beyond about 450 lines. For files exceeding 1,500 lines, F_1 approaches zero.

These results characterize fixing prediction on compact supply-chain instances. A variable's role may be distributed across objective terms, bounds, material-balance equations, and shared-capacity constraints. As LP inputs become longer, prediction requires relating these occurrences across a larger formulation. The observed decline identifies input length as a constraint on reference-fixing recovery in this setting.

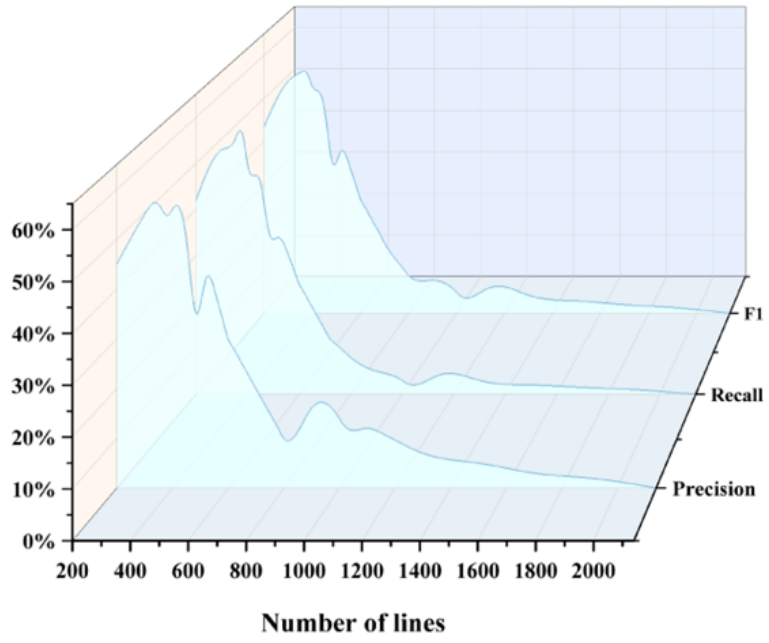

Figure 3: Precision, recall, and F1 for variable-pruning predictions after training on 400 examples. Input size is measured by LP-file line count, and the metrics compare predicted fixings with reference annotations.

6 Discussion and Limitations

Task adaptation increases complete and executable construction outputs, while reference-result matching remains substantially lower than executability. Thus successful execution captures only one aspect of construction quality: the generated objective, domains, and constraints must also express the requested problem. In the workforce case, adaptation introduces explicit resource requirements, period definitions, and operating decisions within the existing constraint organization.

Pruning performance depends strongly on LP input length. The adapter recovers reference fixings most effectively on compact formulations, with decreasing precision and recall on longer inputs. This pattern is consistent with the need to relate each variable's occurrences across objective terms, bounds, balance equations, and shared-capacity constraints.

7 Conclusion

EvoOpt-LLM uses independent task adapters to construct formulations and code, extend LP models with business rules, and predict candidate zero fixings. The experiments show improved construction with adaptation, more explicit workforce-response structure, and partial recovery of fixing annotations with declining performance on longer LP inputs. Together, these results characterize task-specific adaptation for constructing and updating supply-chain optimization models, with numerical solvers determining the remaining decisions.

AI USE STATEMENT

Generative AI tools assisted language polishing and manuscript preparation. LLMs are also a research component of this work: their adaptation and use for model construction, constraint injection, and variable pruning are described in Sections 4 and 5.1. The authors take responsibility for the final content.

REPRODUCIBILITY STATEMENT

Section 5.1 specifies the backbone, hardware, data sizes, and inference interface. Sections 5.2 and 5.4 define the evaluation metrics. Appendix A gives business-rule templates; Appendix B records the environment, adapter-training configuration, and loss masking. Appendix C describes the task datasets, compact-instance preparation, and evaluation procedure.

REFERENCES

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Optimization modeling using MIP solvers and large language models. *arXiv preprint arXiv:2310.06116*, 2023. URL https://arxiv.org/abs/2310.06116.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. URL https://arxiv.org/abs/2108.07732.

Dimitris Bertsimas and Bartolomeo Stellato. The voice of optimization. *Machine Learning*, 110(2): 249–277, 2021.

Hanting Chen, Yasheng Wang, Kai Han, et al. Pangu Embedded: An efficient dual-system LLM reasoner with metacognition. *arXiv preprint arXiv:2505.22375*, 2025. URL https://arxiv.org/abs/2505.22375.

Z. Ding, Z. Tan, J. Zhang, and T. Chen. OR-R1: Automating modeling and solving of operations research optimization problem via test-time reinforcement learning. *arXiv preprint arXiv:2511.09092*, 2025. URL https://arxiv.org/abs/2511.09092.

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 *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2106.09685.

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*, 2025. 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. Mamo: A mathematical modeling benchmark with solvers. *arXiv preprint arXiv:2405.13144v1*, 2024. URL https://arxiv.org/abs/2405.13144v1.

C. Jiang, X. Shu, H. Qian, X. Lu, J. Zhou, A. Zhou, and Y. Yu. LLMOPT: Learning to define and solve general optimization problems from scratch. *arXiv preprint arXiv:2410.13213*, 2024. URL https://arxiv.org/abs/2410.13213.

Jiajun Li, Yixuan Li, Ran Hou, Yu Ding, Shisi Guan, Jiahui Duan, Xiongwei Han, Tao Zhong, Vincent Chau, Weiwei Wu, Zhiyuan Liu, and Wanyuan Wang. Constraint matters: Multi-modal representation for reducing mixed-integer linear programming. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2508.18742.

S. Li, J. Kulkarni, I. Menache, C. Wu, and B. Li. Towards foundation models for mixed integer linear programming. *arXiv preprint arXiv:2410.08288*, 2024. URL https://arxiv.org/abs/2410.08288.

F. Liu, X. Tong, M. Yuan, X. Lin, F. Luo, Z. Wang, et al. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *arXiv preprint arXiv:2401.02051*, 2024. URL https://arxiv.org/abs/2401.02051.

H. Lu, Z. Xie, Y. Wu, C. Ren, Y. Chen, and Z. Wen. OptMATH: A scalable bidirectional data synthesis framework for optimization modeling. *arXiv preprint arXiv:2502.11102*, 2025. URL https://arxiv.org/abs/2502.11102.

Zeyuan Ma, Yue-Jiao Gong, Hongshu Guo, Jiacheng Chen, Yining Ma, Zhiguang Cao, and Jun Zhang. LLaMOCO: Instruction tuning of large language models for optimization code generation. *IEEE Transactions on Evolutionary Computation*, 2026.

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, Ravichandra Addanki, Tharindi Hapuarachchi, Thomas Keck, James Keeling, Pushmeet Kohli, Ira Ktena, Yujia Li, Oriol Vinyals, and Yori Zwols. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2021. URL https://arxiv.org/abs/2012.13349.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, et al. NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 Competition Track*, pp. 189–203. PMLR, 2023.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

R. Thind, Y. Sun, L. Liang, and H. Yang. OptimAI: Optimization from natural language using LLM-powered AI agents. *arXiv preprint arXiv:2504.16918*, 2025. URL https://arxiv.org/abs/2504.16918.

Niki van Stein and Thomas Bäck. LLaMEA: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29 (2):331–345, 2025. doi: 10.1109/TEVC.2024.3497793.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

Z. Xiao, D. Zhang, Y. Wu, L. Xu, Y. J. Wang, X. Han, et al. Chain-of-experts: When LLMs meet complex operations research problems. In *International Conference on Learning Representations*, 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822, 2023.

H. Ye, J. Wang, Z. Cao, F. Berto, C. Hua, H. Kim, et al. ReEvo: Large language models as hyperheuristics with reflective evolution. In *Advances in Neural Information Processing Systems*, volume 37, pp. 43571–43608, 2024.

Haofeng Yuan, Jianing Peng, Jieyi Bi, Ni Zhang, Shiji Song, and Zhiguang Cao. FormuEvo: LLM-guided evolution for discovering solver-efficient mixed-integer programming formulations. *arXiv preprint arXiv:2608.23353*, 2026. URL https://arxiv.org/abs/2608.23353.

A CONSTRAINT FAMILIES AND NOTATION

This appendix presents mathematical templates for the business-rule families in Section 4.3. Table 4 lists the variables and parameters. Each template specifies the added restrictions and, where needed, the constraints linking auxiliary decisions to existing model variables. Instantiating a template requires the input model's applicable variable names and index sets, together with numerical business parameters. The original objective, balance equations, and other constraints remain part of the augmented model. Numerical values in the examples illustrate individual instances; parameter variation across the training examples is described in Appendix C. Indices i, m, and t denote items or orders, machines, and periods.

Table 4: Variables and parameters for the business-rule templates. Subscripts identify the applicable item, machine, and time period.

Symbol	Domain	Meaning
$x_{i,m,t}$	Nonnegative continuous	Production quantity
$s_{m,t}$	Binary	Machine in preparation state
$y_{m,t}$	Binary	Machine producing/operating
$z_{m,t}$	Binary	Start of a new production run
r_m, R_t	Nonnegative parameters	Workforce requirement and availability
$k_{i,t}$	Nonnegative integer	Number of quantity units/batches
$u_{i,t}$	Binary	Production or procurement activated
$q_{i,t}$	Nonnegative continuous	Production or procurement quantity
$I_{i,t}$	Nonnegative continuous	Inventory
$d_{i,t}$	Nonnegative continuous	Delivery quantity
$v_{i,t}$	Binary	Delivery activated

A.1 SETUP TIME

Let L denote the preparation duration. Production is linked to the production state, and setup and production cannot occur simultaneously:

$$0 \le x_{i,m,t} \le U_{i,m,t} y_{m,t}, \qquad y_{m,t} + s_{m,t} \le 1. \tag{14}$$

A production-start trigger satisfies

$$z_{m,t} \ge y_{m,t} - y_{m,t-1}. \tag{15}$$

To require preparation *before* this start, a temporally consistent duration constraint is

$$\sum_{\tau=t-L}^{t-1} s_{m,\tau} \ge L z_{m,t}. \tag{16}$$

For the three-period example, $L = 3$. Initial production states and any preparation completed before the horizon must be provided, or starts without the required history must be excluded. Equation 16 indexes the preceding periods because $z_{m,t}$ marks the first production period.

A.2 WORKFORCE LIMITS

For the time-indexed illustrative model, the workforce block is

$$\sum_{m} r_m y_{m,t} \le R_t, \qquad 0 \le x_{i,m,t} \le U_{i,m,t} y_{m,t}. \tag{17}$$

Here $U_{i,m,t}$ replaces a generic big-M with an instance-valid upper bound. If a formulation distinguishes an operating state from a production state, their linking constraints must be specified. The workforce definition must also determine whether preparation consumes labor; the displayed block accounts for the operating state represented by $y_{m,t}$.

A.3 Batch quantities and inventory capacity

The example business rules specify a quantity multiple of 100, a minimum activated quantity of 1,000, and an inventory capacity:

$$q_{i,t} = 100k_{i,t}, \qquad 1000u_{i,t} \le q_{i,t} \le U_{i,t}u_{i,t}, \tag{18}$$

$$0 \le I_{i,t} \le I_i^{\max}. \tag{19}$$

The batch multiplier is a nonnegative integer and activation is binary. The values describe this example; training records use varying business parameters. Inventory balance equations from the original model continue to link inventory with production, procurement, and demand.

A.4 Production across periods

Time-indexed production is constrained by machine capacity and total demand:

$$\sum_i x_{i,m,t} \le \mathrm{Cap}_{m,t}, \qquad \sum_{m,t} x_{i,m,t} = D_i. \tag{20}$$

This quantity-capacity expression assumes compatible production units; a time-capacity model instead needs item-specific processing coefficients. The demand equation is the simplified template in which production directly satisfies demand. Bill-of-materials balances, substitutions, or initial inventory must remain represented where they are part of the original formulation. These constraints alone do not override a pre-existing continuity restriction.

A.5 Split deliveries

For demand D_i, delivery bounds $d_i^{\min}$ and $d_i^{\max}$, and minimum spacing Δ, the template is

$$\sum_t d_{i,t} = D_i, \qquad d_i^{\min} v_{i,t} \le d_{i,t} \le d_i^{\max} v_{i,t}, \tag{21}$$

$$\sum_{\tau=t}^{t+\Delta-1} v_{i,\tau} \le 1. \tag{22}$$

Spacing constraints apply to valid windows in the time horizon. If deliveries must be quantity multiples, the corresponding integer batch relation from Equation 18 is also required. Allowing multiple deliveries is compatible with pure augmentation only when an existing single-delivery restriction does not forbid them.

B Experimental environment and training configuration

Experiments use an Ascend runtime environment. Transformers provides the inference interface for automated modeling and LP generation. The environment also includes vLLM/vLLM-Ascend and OpenCompass.

Training implementation. The adapter-training configuration targets the attention query, key, value, and output projections. Its LoRA multiplier is $\alpha/r = 64/32 = 2$, absorbed into the factors in Equation 5. With four data-parallel devices, batch size one and eight accumulation steps give a nominal batch of 32 examples per optimizer update.

For instruction–response pairs, preprocessing concatenates the prompt and response, inserts a separating space when necessary, appends the tokenizer's end-of-sequence token, and truncates the combined sequence. Prompt and padding labels are masked with -100. The causal loss shifts the labels for next-token prediction and is normalized over non-ignored targets. With gradient accumulation, Transformers 4.53.2 can normalize the summed loss using the valid-token count for the accumulation group. Equation 6 expresses this token normalization over the response positions retained by preprocessing.

Table 5: Experimental environment and adapter-training configuration.

Component	Configuration
Foundation model	openPangu-Embedded-7B-V1.1
Accelerators	$4\times$ Huawei Ascend 910B2
Memory	64 GB HBM per accelerator
Firmware/driver	$\geq$ 23.0.6
CANN	8.1.rc1
PyTorch	2.5.1
`torch_npu`	2.5.1.post1
Transformers	4.53.2
vLLM	0.9.2
vLLM-Ascend	0.9.2rc1
OpenCompass	0.5.0
Training parameter	**Value**
LoRA rank / alpha / dropout	32 / 64 / 0.05
Target projections	`q_proj`, `k_proj`, `v_proj`, `o_proj`
Learning rate / weight decay	2×10^{-5} / 0.01
Batch size per device / accumulation	1 / 8 steps
Training epochs / seed	3 / 42
Maximum sequence length	20,000 tokens
Schedule / warmup ratio	Linear / 0.03
Precision / gradient checkpointing	BF16 / enabled
Label smoothing	0

C Data construction and evaluation details

Task records. The datasets provide instruction–response pairs for the objective in Section 4.1. A construction pair contains a problem description and a response with its formulation and implementation code. An injection pair contains an existing LP model and an additional business-rule request, followed by a reference LP response. A pruning pair contains the current LP model and a JSON list of zero assignments.

Constraint Injection contains 3,000 examples of business-rule updates. LP-VarPrune contains 413 LP–fixing pairs, with 400 examples used for training in Section 5.1.

Each example stores its instruction and task input in a `prompt` string and its supervised response in a `completion` string. In LP-VarPrune, the response string encodes a JSON array of zero assignments. An anonymous instance identifier accompanies each pruning example as metadata; training uses the prompt and response.

For example, a pruning response for LP variables x_1 and x_2 is serialized as follows:

```
{
  "prompt": "<task instruction and current LP text>",
  "completion": "[\"x_1=0\", \"x_2=0\"]"
}
```

Table 6 summarizes dataset size and exact-string uniqueness. Across the 413 pruning examples, all responses encode zero-assignment lists whose variable names occur in their associated LP inputs. The lists contain 91–4,236 assignments, with a median of 440.

Construction evaluation. The ten training subsets contain 300, 600, 900, 1,200, 1,500, 1,800, 2,100, 2,400, 2,700, and 3,000 examples. A separate adapter is trained for each subset, and all variants use the same held-out test set. A complete nonempty Python block counts toward N_g. The evaluator appends solver-status and objective reporting and executes the block with a timeout. Normal process termination counts toward N_e.

Table 6: Task dataset statistics. Unique prompts and instruction–response pairs are counted by exact string matching.

Dataset	Task	Examples	Unique prompts	Unique pairs
Constraint Injection	Injection	3,000	1,325	1,335
LP-VarPrune	Pruning	413	412	412

For a numerical reference, matching uses the objective $\widehat{z}$ reported when COPT returns `OPTIMAL`. Let z^{ref} be the reference objective and define $\bar{z} = \mathrm{round}(\widehat{z})$ and $\bar{z}^{\mathrm{ref}} = \mathrm{round}(z^{\mathrm{ref}})$, with half-integer ties rounded to the nearest even integer. The numerical acceptance condition is

$$|\bar{z} - \bar{z}^{\mathrm{ref}}| \le \begin{cases} 0.05\,|\bar{z}^{\mathrm{ref}}|, & \bar{z}^{\mathrm{ref}} \neq 0, \\ 0.05, & \bar{z}^{\mathrm{ref}} = 0. \end{cases} \tag{23}$$

Feasibility is assessed by the solver for the generated formulation. Matching uses the objective value, so different optimal decision vectors with the same objective receive the same score.

A returned status other than `OPTIMAL` is encoded as `No Best Solution`; this outcome matches a reference carrying the same label. Missing code, execution errors, timeouts, absent or non-finite numerical outputs, and objective differences exceeding Equation 23 count as unmatched. Each matching output contributes one to N_a in Equation 12.

The all-input execution and reference-matching rates are GE and GA, respectively. At the largest training setting,

$$0.91 \times 0.659 = 0.59969, \qquad 0.91 \times 0.2637 = 0.239967.$$

These values give the approximately 60.0% and 24.0% all-input rates reported in Section 5.2.

Response processing. Each task uses a task-specific adapter and prompt to generate one response per input. The input length and generation budget are checked against the model's context limit. Injection responses are extracted from the objective declaration through the LP closing marker. Pruning responses are parsed as JSON lists of zero assignments, with duplicate entries removed before set-based evaluation.

Injection examples. The base formulation represents demand satisfaction, production capacity, transportation balance, distribution-center capacity, and business logic such as mutual exclusivity. Each editing record combines the existing LP model with a business-rule request and an augmented target.

The five rule families contribute 600 records each. Their parameters vary across examples, including setup durations of 3–12 days, big-M values of 1,000–2,900, and planning horizons of 5–30 days, together with different workforce, batch, inventory, and delivery settings. These parameter ranges provide variation within each family of training examples.

C.1 Compact supply-chain instances for pruning

The compact-instance generator selects a subset of demand items and retains their associated supply-chain relationships. Its configuration specifies a sampling rate of 0.01, a sampling seed of 42, and dates from November 1 through December 31, 2024. The seed controls item sampling, rather than adapter training.

The procedure first identifies demand-item identifiers shared across the readable demand tables in the configured date range. It samples one set of these identifiers and reuses that set as the demand roots for each date. For each daily instance, recursive traversal expands the selected roots through BOM relationships and production alternatives. A subsequent pass adds delivery substitutes through a component-to-alternate mapping. The demand roots are shared across dates, while their retained descendants depend on the relationships present in each daily instance.

The retained items determine the relevant normal and alternative resource groups. Variables, constraint rows, and associated data tables are filtered using item and resource identifiers. Pair con-

straints are retained when both referenced items remain in the selected scope. Calendar and index-mapping information is preserved.

Resource scaling factors are computed from the ratio of retained to original distinct item–resource associations and applied through a resource-code lookup to resource-balance right-hand sides. Filtering and capacity rescaling define a new planning instance rather than an equivalent representation of the full model.

The resulting compact model is exported in LP format, preserving the retained variable identifiers, objective coefficients, constraint rows, bounds, and domains.

Reference-label construction. The pruning labels are extracted from precomputed reference solutions with an optimal solve status. Let $x^\star$ denote the recorded optimal solution of the associated optimization model and ϵ_0 the zero-value tolerance. The reference index set is

$$\mathcal{J}^{\mathrm{ref}} = \{j : |x_j^\star| \leq \epsilon_0\}. \tag{24}$$

The selected variables are mapped to their LP identifiers and serialized as `"variable_name=0"` assignments. Each LP–response pair thus represents the zero pattern of its recorded optimal solution. When multiple optimal solutions exist, that stored solution supplies the reference pattern for training and for the precision, recall, and F_1 comparisons in Section 5.4.

D CONSTRAINT-INJECTION EXAMPLE

Figures 4 and 5 present the workforce response excerpts discussed in Section 5.3. The left panels show input-model excerpts and the right panels show response excerpts. The comparison highlights the adapted response's explicit workforce requirements, period definitions, and operating-variable declarations within the original constraint organization. Equation 17 gives the associated workforce-capacity and production-linking template.

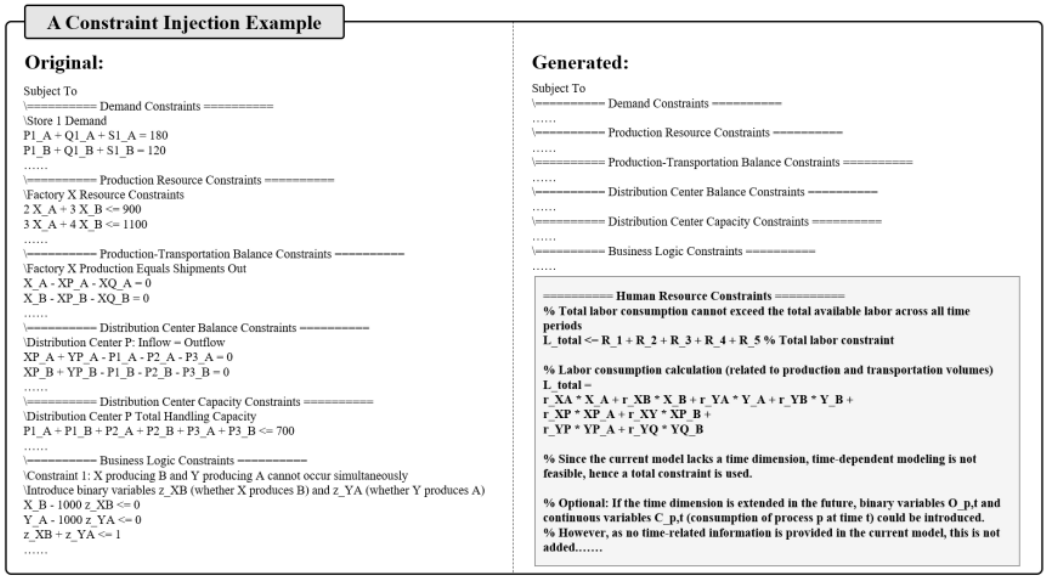

Figure 4: Workforce-update response excerpts from the untuned base model. The response uses aggregate labor expressions and notes the absence of an explicit time dimension in the input.

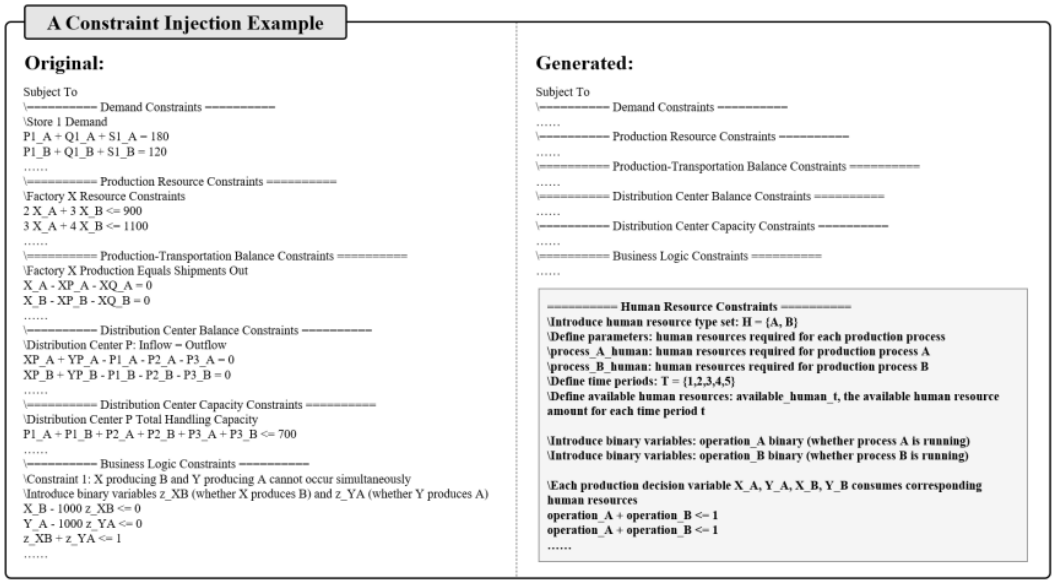

Figure 5: Workforce-update response excerpts after task-specific adaptation. The response introduces resource requirements, period definitions, and operating-variable declarations within the existing constraint organization.