POST-TRAINING OPEN MULTIMODAL MODELS FOR OPTIMIZATION MODELING FROM TEXT AND IMAGES

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have advanced automated optimization modeling, but existing methods largely assume that problems are fully specified in text. In practice, text and images often provide complementary information needed to formulate a problem. While existing multimodal optimization modeling benchmarks reveal a substantial performance gap between open and proprietary multimodal large language models (MLLMs), they do not provide training data or post-training methods for this task. We therefore ask whether task-specific post-training can equip a compact open MLLM with this capability. We introduce MLLM4OR-Corpus, comprising 22,107 solver-verified instances. We partition the corpus into disjoint training, development, and test subsets for post-training, model selection, and in-domain evaluation, respectively. For external evaluation, we use MM-OptBench and introduce LLM4OR-VL to cover a wider range of problem sources and description styles. LLM4OR-VL comprises 878 multimodal problems reconstructed from eight text-only optimization benchmarks. Using these resources, we introduce MLLM4OR, an open 9B-scale multimodal LLM trained to generate mathematical formulations and corresponding solver code from multimodal specifications. To this end, we first use supervised fine-tuning (SFT) to connect text-image evidence to formulations and solver code, comparing training answers with no explanations, concise explanations, or detailed explanations of the modeling and implementation steps. We then propose a solver-guided preference construction method for direct preference optimization (DPO), contrasting verified answers with the model's own execution and modeling failures. Extensive experiments show that including concise explanations for some training problems and omitting them for others yields the strongest overall SFT performance. DPO further improves overall accuracy and performance on MM-OptBench and LLM4OR-VL. These findings support task-specific post-training for automated multimodal optimization modeling with compact open MLLMs.

1 INTRODUCTION

In operations research, optimization modeling develops a mathematical formulation from an informal problem specification and implements it as a solver-executable program (Fourer et al., 1990). The formulation defines the index sets, decision variables, objective, and constraints that characterize the problem. These models support decision-making across logistics (Dantzig & Ramser, 1959), supply chains (Shah, 2005), energy (Padhy, 2004), health care (Gupta & Denton, 2008), aviation (Barnhart et al., 2003), and finance (Markowitz, 1952). Much of the difficulty arises before solver execution: the specification, formulation, and implementation must describe the same problem (Willemain, 1995). A missing constraint, unit mismatch, or indexing error can silently alter the problem being solved even when the code runs normally. Automating this process could reduce formulation and verification effort and lower the expertise barrier to using optimization in practice (Xiao et al., 2025).

LLMs are increasingly used to automate optimization modeling and have shown encouraging results. Broadly, existing LLM-based methods fall into three categories. Prompting and agentic approaches use structured prompts or coordinated LLM calls to divide the workflow into formulation, code generation, execution, and repair, as in Chain-of-Experts (Xiao et al., 2024) and OptiMUS (AhmadiTeshnizi et al., 2024). Supervised fine-tuning (SFT) approaches train models on natural-language problem specifications paired with target formulations and solver code, as in ORLM (Huang et al., 2025) and

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.

OptMATH (Lu et al., 2025). Reinforcement learning (RL) and process-supervised approaches instead use solver feedback, intermediate modeling signals, or multi-turn interaction traces, as in SIRL (Chen et al., 2025), MURKA (Xie et al., 2025), StepORLM (Zhou et al., 2026), and PEARL (Lu et al., 2026). Despite these differences, all assume that the optimization problem is fully specified in text.

However, many optimization problems are not specified through text alone. Text may describe the objective and operating rules, while visual inputs convey the numerical values and spatial, network, or temporal structure needed for formulation. For example, in routing and urban logistics (see Fig. 10(a)), decision-support systems combine maps and road-network views with traffic, demand, capacity, and access restrictions (Ioannou et al., 2002; Santos et al., 2011). Production-planning systems (see Fig. 10(b)) present product dependencies, resource states, and schedules visually (Sun et al., 2020). Energy-system interfaces (see Fig. 10(c)) likewise expose process configurations, grid states, time series, and contingency scenarios that inform design and operational decisions (Granacher et al., 2022; Wong et al., 2014). This use of visual information is longstanding: graphical modeling and interactive optimization systems support model construction and inspection through diagrams and linked data or solver interfaces (Chari & Sen, 1998; Pirkul et al., 1999). A text-only system could instead require users to restate all relevant information conveyed visually in a self-contained textual specification. Preparing that specification, however, is itself part of the modeling work: it requires identifying every relevant quantity and entity, together with the relations and structural dependencies among them. Manual transcription can omit relevant information, assign values to the wrong entities, or misrepresent spatial, temporal, and network relationships. Direct multimodal optimization modeling is therefore essential for end-to-end automation: otherwise, users must still interpret the visual information and translate it into text themselves.

Automating this manual step requires *multimodal optimization modeling*: jointly understanding text and images to generate a mathematical formulation and corresponding solver code. IntentOpt (Ahmed et al., 2026) and MM-OptBench (Li et al., 2026b) reveal a substantial performance gap between open and proprietary MLLMs. On MM-OptBench, even the strongest proprietary models solve only about half the instances, while the evaluated open multimodal-mathematics specialists fail almost entirely, including when visual information is supplied in text. These findings expose difficulties in both joint text-image understanding and downstream mathematical reasoning. Both studies focus on evaluation; neither develops training data or post-training methods for this task. We therefore ask whether targeted SFT and preference optimization can equip a compact open MLLM with this capability. We focus on open models for their promising multimodal performance (Bai et al., 2025) and accessible weights, which enable reproducible, task-specific adaptation (Deitke et al., 2024).

Training open MLLMs for multimodal optimization modeling requires instances that include a text-image problem specification, a corresponding mathematical formulation, and executable solver code. To our knowledge, no public training corpus has been developed for this purpose. We therefore introduce MLLM4OR-Corpus, comprising 22,107 solver-verified instances constructed using three complementary sources: OptMATH-Train (Lu et al., 2025), OR-Instruct (Huang et al., 2025), and new problems sampled from parametric generators. We partition the corpus into three disjoint subsets: MLLM4OR-Train for model training via SFT and preference optimization, MLLM4OR-Dev for model selection, and MLLM4OR-Test for in-domain evaluation. For external evaluation, we use MM-OptBench (Li et al., 2026b) and introduce LLM4OR-VL to cover a wider range of problem sources and description styles. LLM4OR-VL comprises 878 multimodal instances built using problems from eight text-only optimization benchmarks as source material: ComplexOR (Xiao et al., 2024), IndustryOR (Huang et al., 2025), MAMO Complex LP and MAMO Easy LP (Huang et al., 2024), NL4Opt (Ramamonjison et al., 2023), NLP4LP (AhmadiTeshnizi et al., 2024), OptMATH-Bench (Lu et al., 2025), and ReSocratic (Yang et al., 2025). We design complementary text-image inputs for these problems and provide private reference answers for evaluation.

Using these resources, we develop MLLM4OR by post-training Qwen3.5-9B (Qwen Team, 2026) for multimodal optimization modeling. SFT teaches the model to generate formulations and solver code from text-image inputs, while subsequent preference optimization uses solver feedback to refine the policy. We compare four SFT variants: Direct-SFT, Compact-SFT, Detailed-SFT, and Mixture-SFT. For Direct-SFT, we train on formulations and solver code alone. For Compact-SFT and Detailed-SFT, we add concise or expanded chain-of-thought (CoT) explanations of the modeling and implementation steps. For Mixture-SFT, we combine targets with and without concise explanations in controlled proportions. All settings use the same training problems; reference-answer preparation is described in §E.1. We develop solver-grounded direct preference optimization (SG-DPO) (Rafailov

et al., 2023) with error-stratified preference construction. We pair verified answers with the model's own failures on the same text-image problem, distinguishing programs that report an optimum but return the wrong objective value from basic format and execution failures.

We evaluate MLLM4OR on the in-domain MLLM4OR-Test and two external benchmarks, LLM4OR-VL and MM-OptBench, alongside general-purpose and mathematical-reasoning MLLMs. Three findings emerge. 1) *Task-specific supervision is effective:* SFT substantially improves formulation-and-code generation over the untuned baseline. 2) *More explanation is not always better:* mixing training answers with and without concise explanations yields the strongest aggregate SFT performance. 3) *Solver feedback benefits external performance:* our DPO comparison (§4) shows higher mean accuracy on both external benchmarks and better aggregate accuracy and four-sample coverage, despite a small in-domain decrease. The gains span most external sources, with lower across-run variability.

Our contributions are fourfold. 1) To our knowledge, we present the first systematic study of post-training an open MLLM to generate mathematical formulations and solver code from problems jointly specified by text and images. **2)** We construct MLLM4OR-Corpus, a solver-verified corpus with disjoint training, development, and test splits, and LLM4OR-VL, a multimodal test suite reconstructed from eight text-only optimization benchmarks. **3)** We develop selective and mixed reasoning supervision for SFT, together with solver-grounded DPO that targets execution failures and modeling errors in executable programs. **4)** We provide execution-grounded evaluations, preserving generated answers and execution evidence for analyses of supervision, preference learning, and multi-sample coverage.

2 Related Work

Automated optimization modeling. Existing natural-language-to-optimization (NL-to-Opt) methods broadly follow three routes. Prompting and agentic methods coordinate LLM calls to formulate, implement, and revise models, as in Chain-of-Experts (Xiao et al., 2024) and OptiMUS (AhmadiTeshnizi et al., 2024). Supervised fine-tuning (SFT) trains models on problem descriptions paired with target formulations or solver code, as in ORLM (Huang et al., 2025) and OptMATH (Lu et al., 2025). Reinforcement learning (RL) and process supervision use solver outcomes or intermediate modeling feedback to improve generation, as in Solver-Informed RL (Chen et al., 2025), MURKA (Xie et al., 2025), StepORLM (Zhou et al., 2026), and PEARL (Lu et al., 2026). OptArgus (Li et al., 2026a) complements generation with multi-agent consistency checks across the problem description, formulation, and code. The accompanying benchmark ecosystem now spans NL4Opt (Ramamonjison et al., 2023), ComplexOR (Xiao et al., 2024), IndustryOR (Huang et al., 2025), MAMO (Huang et al., 2024), ReSocratic/OptiBench (Yang et al., 2025), OptMATH-Bench (Lu et al., 2025), Bench4Opt (Wang et al., 2026), and the industrial-scale MIPLIB-NL (Li et al., 2026c). Despite these advances, existing methods assume that the optimization problem is fully specified by text alone. IntentOpt (Ahmed et al., 2026) and MM-OptBench (Li et al., 2026b) extend evaluation to multimodal problem specifications, but focus on assessing existing models rather than post-training open MLLMs to generate both mathematical formulations and solver code. Thus, a learning method for end-to-end multimodal optimization modeling remains missing.

Multimodal mathematical reasoning. MathVista (Lu et al., 2024) and MathVerse (Zhang et al., 2024a) evaluate mathematical reasoning from text and images. Representative post-training approaches include curated and synthetic instruction data in Math-LLaVA (Shi et al., 2024), image-to-rendering-code supervision in MathCoder-VL (Wang et al., 2025), and outcome- and process-level feedback in MM-Eureka (Meng et al., 2025) and MM-PRM (Du et al., 2025), respectively. These methods primarily target answers and reasoning traces. Multimodal optimization modeling requires a different combination of capabilities: interpreting complementary text-image evidence, applying real-world domain knowledge, formalizing operational requirements as decision variables, objectives, and constraints, and implementing the formulation in solver code. In practical applications, optimization models can involve thousands to hundreds of thousands of variables (Li et al., 2026c), making manual solution impractical. The goal is therefore to construct a faithful model that a solver can optimize, not merely produce a numerical answer. MLLM4OR addresses this integrated task through supervision on mathematical formulations and solver code, together with solver-grounded feedback. §A provides the extended comparison.

3 MLLM4OR: Data Construction and Post-Training

We define multimodal optimization modeling (§3.1), construct the solver-verified MLLM4OR-Corpus for training, model selection, and evaluation (§3.2) and LLM4OR-VL for broader evaluation (§3.3), and present selective and mixed reasoning supervision for SFT and solver-grounded DPO (§3.4).

3.1 Problem Statement

Definition 1 (Optimization modeling)**.** Optimization modeling is the task of translating a problem specification into a mathematical formulation m that defines the decision variables and their domains, the objective, and the constraints. The formulation must faithfully represent the entities, numerical data, and operational requirements in the specification.

Definition 2 (Multimodal optimization modeling)**.** Given an input $x = (I, t)$, where I denotes the visual input (one or more images) and t is complementary text, multimodal optimization modeling produces a response $y = [m; c]$ containing a mathematical formulation m and its solver-executable implementation c. The formulation m must capture the decisions, objective, and constraints specified jointly by I and t, and c must implement m using the same instance data. Neither modality is assumed sufficient on its own. An MLLM with parameters θ generates the response according to

$$y = [m; c] \sim \pi_\theta(\cdot \mid x). \tag{1}$$

Equivalent formulations are admissible provided that they preserve the feasible decisions and objective specified by x. The task is to construct m and c, not to predict an optimal decision or objective value directly; a solver computes the solution by executing c.

3.2 Constructing MLLM4OR-Corpus for Multimodal Optimization Modeling

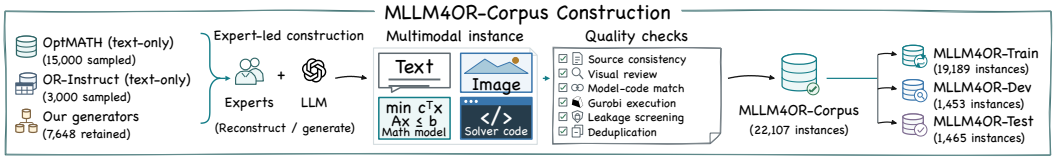


Figure 1: MLLM4OR-Corpus construction. With expert guidance and LLM assistance, we reconstruct multimodal instances from text-only problems in OptMATH and OR-Instruct, and design generators to create new instances. Both routes produce text, an image, a mathematical formulation, and solver code. After rigorous quality control, including expert review, Gurobi execution, and deduplication, we split the 22,107 retained instances into disjoint training, development, and test sets.

To our knowledge, no public training corpus supports learning to generate mathematical formulations and solver code from complementary text-image inputs. Established training corpora are text-only (Huang et al., 2025; Yang et al., 2025; Lu et al., 2025), while multimodal benchmarks provide evaluation instances without dedicated training sets (Ahmed et al., 2026; Li et al., 2026b). We construct MLLM4OR-Corpus to fill this gap. Each instance pairs an input $x = (I, t)$ with a verified reference formulation $m^\star$ and an executable solver program $c^\star$. The text and image jointly specify the problem data and requirements. Disjoint training, development, and test partitions support model training, model selection, and in-domain evaluation, respectively.

Source selection. The three sources provide complementary coverage. OptMATH-Train supplies diverse optimization structures (Lu et al., 2025); we select 15,000 candidates using visual suitability, family quotas, and per-group sampling limits (§C.2; Figure 6). We include OR-Instruct (Huang et al., 2025) for its industry-inspired scenarios and varied descriptions of business objectives and operational rules, and audit its full 3,000-record release (§C.3; Figure 7). Converting existing problems leaves their sizes and structures fixed. Our parametric generators vary problem size and constraint structure across six optimization problem families, yielding 7,648 training instances (§C.4; Figure 8).

Multimodal instance construction. Human experts lead problem specification, text–image design, and semantic validation; MLLMs assist with drafting and refinement. Images are generated by Python rendering programs from structured problem data. For problems from *OptMATH-Train*, we remove hints about the formulation or code and use the original data to draw labeled matrices, networks, or charts. We rewrite the text to state what must be decided and any rules not shown in the image (§C.2). For *OR-Instruct*, we first check the business requirements and numerical data for omissions or inconsistencies. We then draw tables, timelines, or diagrams from the checked data and rewrite the accompanying text (§C.3). Our *parametric generators* instead create new problems by sampling their structures and numerical values. From each sampled problem, we produce the text, image, formulation, and solver code, with images drawn as networks, schedules, or plots (§C.4; Figure 9). In all three cases, experts check that the images show the correct values and relationships, have readable labels, and agree with the text. They request revisions to unclear or misleading visuals and check that the formulation and solver code represent the same problem.

Quality assurance. Experts review the problem specification, text–image input, formulation, and code together with automated check results. This process covers source consistency, visual readability, formulation–code agreement, Gurobi execution, leakage screening, and deduplication (Figure 1). Identified issues are corrected and rechecked; instances with unresolved substantive problems are excluded. Successful execution alone does not show that the formulation and code capture the intended problem. §C.1 details the review team, assessment criteria, correction and adjudication procedures, and estimated workload. §C.5 describes the automated input checks and Gurobi execution protocol, including their scope and limitations.

Corpus composition and scale. The resulting MLLM4OR-Corpus contains 22,107 text–image problems: 19,189 for training, 1,453 for development, and 1,465 for testing (§C.6–§C.7). Figure 2(a) shows application contexts ranging from logistics and industrial operations to energy, finance, and health, alongside unspecified or mixed settings. The task distribution (b) is led by network and routing problems (30.3%), followed by production and resources (18.3%) and scheduling and staffing (16.2%). Binary and mixed-integer linear models account for 69.5% of the reference formulations (c). Panels (d,e) show that model sizes range from small formulations to ones with thousands of variables or constraints; the medians are 35 variables and 24 constraints (§C.8).

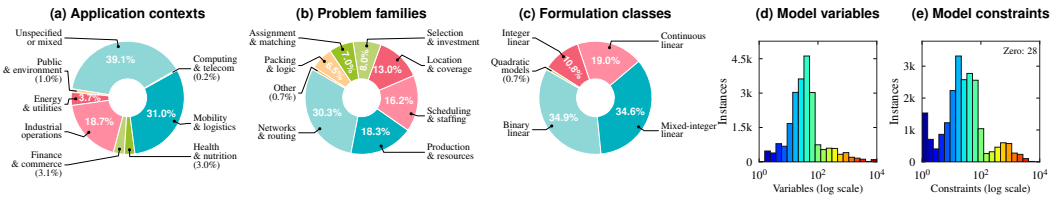


Figure 2: Composition and reference-model sizes of MLLM4OR-Corpus (22,107 instances).

3.3 Evaluation Data Design and Construction

We evaluate in-domain generalization on MLLM4OR-Test, the 1,465-instance held-out partition of MLLM4OR-Corpus. To assess performance on other problem sources and visual presentations, we also use 780 problems from MM-OptBench (Li et al., 2026b) that require the same mathematical-formulation and solver-code outputs (§D.2). We further construct LLM4OR-VL, an evaluation suite of 878 text–image problems reconstructed from eight text-only sources (§D.3).

Constructing LLM4OR-VL**.** We use problems from eight text-only datasets: ComplexOR, IndustryOR, MAMO Easy LP, MAMO Complex LP, NLP4LP, NL4Opt, ReSocratic, and OptMATH-Bench. We select problems spanning different structures and scenarios and remove duplicates, retaining 878 of 1,677 source records. Figure 11 outlines the construction and quality-control workflow (§D.3 details source selection, duplicate removal, and the retained counts). Experts prepare a self-contained specification for each selected problem. We render its data and relations as diagrams, tables, or charts, with accompanying text stating the decision task and remaining rules (§D.4 explains source

corrections and how information is divided between text and image; Figure 10(g)–(i) gives examples). Quality control combines expert review of source consistency, visual readability, reference answers, and answer leakage with automated file and input-binding checks. Identified issues are corrected and rechecked before inclusion; §D.5 details this procedure and its estimated workload. The input profile in Figure 12 (§D.7) shows varied contexts, including logistics, industrial operations, and health care, and visual formats ranging from illustrated boards to tables and charts. This makes established text-only optimization problems available for evaluating models that must combine textual requirements with visual data.

3.4 Post-Training MLLM4OR

We first use SFT to train the model to generate mathematical formulations and solver code from text–image inputs. We study whether adding explanations to the training answers improves learning (§3.4.1). We then use solver feedback to identify mistakes in the model's responses and train it to prefer verified answers through DPO (§3.4.2; Figure 4).

3.4.1 Supervised Fine-Tuning

When a training answer contains only a formulation and solver code, it shows the modeling decisions but not how they follow from the problem description. Adding a chain-of-thought (CoT) explanation (Wei et al., 2022; Zhang et al., 2024b) connects these decisions to the text and image, for example, by explaining why a variable counting vehicles must be integer. However, prior work suggests that longer, more complex explanations may be harder for smaller models to learn from (Jia et al., 2024) (§A.3).

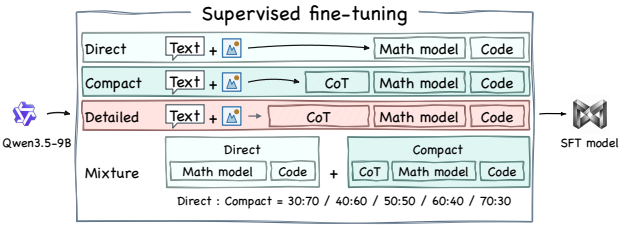


Figure 3: SFT with and without explanations.

We therefore compare four SFT designs on the same 19,189 problems, using verified formulations and solver code as reference outputs (Figure 3). Direct-SFT uses answers containing only the formulation $m^\star$ and code $c^\star$. To test whether explanations improve learning, Compact-SFT adds a concise explanation before them, linking the text and image to sets, parameters, variables, the objective, constraints, and an implementation plan. Detailed-SFT adds finer-grained references to supporting text or image regions and more implementation detail to test whether further detail helps. Mixture-SFT uses concise explanations for some problems and no explanations for the rest; varying this split tests how often explanations should be included in training.

Frontier LLMs generate detailed explanations using the verified formulations as references; we then apply fixed compression rules to obtain concise explanations (§E.1). For Mixture-SFT, we assign each problem a Direct-SFT or Compact-SFT answer according to the chosen mixture ratio before training. This assignment remains fixed throughout training; each problem appears only once in the dataset, not once for each answer format. We compare Direct:Compact ratios of 30:70, 40:60, 50:50, 60:40, and 70:30 by problem count; unless stated otherwise, Mixture-SFT denotes the 70:30 variant. The assignment rules and mixture statistics are given in §E.2 (Table 16).

Given text–image input $x_i = (I_i, t_i)$, format instruction v_i, and reference answer y_i, we minimize the next-token loss $\mathcal{L}_{\mathrm{SFT}}(\theta) = -\sum_{i,j} \log \pi_\theta(y_{ij} \mid x_i, v_i, y_{i,<j})$, where y_{ij} is the j-th answer token and $y_{i,<j}$ its prefix. The loss covers only the reference answer, including any explanation; input text, images, and instructions serve as context. At evaluation, all variants use the same output instruction, requesting only a formulation and solver code in one call.

3.4.2 Solver-Grounded Direct Preference Optimization

During SFT, the model learns from reference solutions without explicitly comparing them with its own mistakes. A generated program may run successfully and return an optimum even if it uses a capacity of 8 where the image specifies 5. An optimal solver result therefore does not guarantee

that the code correctly represents the problem described in the text and image. We use SG-DPO to train the model to prefer a verified reference answer over its own incorrect answer to the same problem. We include programs that run and report an optimum but return the wrong objective value, because execution and solver-status checks alone would miss these errors. Figure 4 illustrates how we construct the training pairs and update the model.

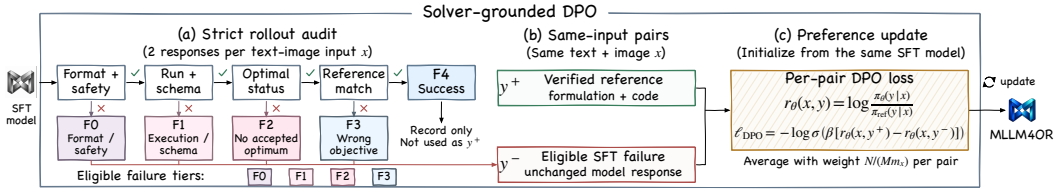


Figure 4: Solver-grounded DPO with problem-normalized pair weights. Eligible failures can contribute multiple pairs per input.

A single pass/fail label does not distinguish a malformed response from an executable program with an incorrect objective. We instead record the stage at which verification fails. For each training input, we sample two responses and check format, execution, solver outcome, and agreement with the same-instance reference. F0 denotes a format or static-safety failure; F1 a syntax, runtime, or result-schema failure; F2 execution without an accepted optimal finite outcome; F3 an accepted optimal outcome with a mismatched objective; and F4 reference agreement. The order in Figure 4(a) follows verification progress, not a universal scale of error severity or a diagnosis of its cause.

We classify eligible failures as static-safe F0, qualified F1, F2, or F3, distinguishing executable but incorrect answers from basic reliability failures. Distinct eligible failures on the same problem can both supply a preference pair; exact or redundant duplicates are removed. We do not discard otherwise eligible pairs merely to enforce source proportions or coarse template quotas. Instead, inverse-multiplicity weights prevent a problem with two retained failures from receiving twice the total weight of a problem with one. §F.4 gives the pair-construction rules.

Each preference compares two responses to the same text and image, so it concerns how to solve one problem rather than differences between problems. For input $x = (I, t)$, we use its canonical verified response as y^+ and an unchanged, eligible policy-generated failure as y^-. Using actual failures targets errors the policy makes; using the reference as the positive allows us to learn from an input even when both sampled responses are wrong. F4 responses are recorded but do not replace the canonical target, since matching an objective value does not guarantee an equivalent formulation. Unsafe programs are never executed, and overlength pairs are deferred intact rather than losing part of a formulation or program to truncation.

The resulting triples form a fixed preference dataset $\mathcal{D}_{\text{pref}}$. We initialize the policy and frozen reference π_{ref} from the same Mixture-SFT checkpoint. Writing $r_\theta(x, y) = \log[\pi_\theta(y \mid x)/\pi_{\text{ref}}(y \mid x)]$, the per-pair DPO loss is $\ell = -\log \sigma\big(\beta[r_\theta(x, y^+) - r_\theta(x, y^-)]\big)$ (Rafailov et al., 2023). We minimize its prompt-normalized average, using multiplier $N/(Mm_x)$ for each pair, where N is the number of pairs, M the number of distinct inputs, and m_x the number of pairs for input x. Our contribution is the construction of task-specific preferences, not a new preference-loss family. Verification precedes training, so updates require neither online solver calls nor a learned reward model. From 2,000 approximately source-balanced training problems and 4,000 responses, the completed run retains 498 pairs from 315 problems. Algorithm 1 gives the procedure; §F.1 illustrates it, and §F.6 discusses the limits of component-level conclusions.

4 Experiments

We address three research questions: the effectiveness of task-specific post-training (RQ1; §4.1), the role of explanations in SFT supervision (RQ2; §4.2), and the additional gains from solver-grounded DPO (RQ3; §4.3). Each subsection answers one question using the shared setup below.

Experimental setup. 1) *Datasets.* We use MLLM4OR-Train (19,189 instances) for training and the complete MLLM4OR-Dev (1,453) for model selection. Evaluation covers MLLM4OR-Test (1,465),

MM-OptBench (780), and LLM4OR-VL (878). 2) *Models.* Our Qwen3.5-9B variants are Direct-SFT, Compact-SFT, Detailed-SFT, Mixture-SFT, and Mixture-SFT-initialized SG-DPO. We compare them with seven general-purpose baselines: Qwen3.5-9B, Qwen3.6-Plus, GLM-4.5V, Kimi-K2.5, Gemini 3.1 Pro, Claude Sonnet 4.6, and GPT-5.5. Table 1 also includes MathCoder-VL-8B (Wang et al., 2025), MM-Eureka (Meng et al., 2025), and MM-PRM (Du et al., 2025) as mathematical baselines. Local experiments use one NVIDIA H200 and multiple NVIDIA H100 GPUs. 3) *Metrics.* We execute generated code and compare its result with the reference answer. Pass@1 measures accuracy with one answer per problem (five-run mean ± SD where available). Pass@4 is the fraction solved by at least one of four samples. Micro pools all problems; macro averages the three test-set accuracies. §B provides training, inference, and evaluation details; §F.6 and §G.7 describe DPO selection and mathematical-baseline protocols, respectively.

4.1 RQ1: How Effective Is Task-Specific Post-Training?

Table 1 compares our post-trained models with the untuned backbone, general-purpose MLLMs, and mathematical-reasoning models.

Table 1: Pass@1 (%). Subscripts: five-run sample SD. Bold: column best in the Qwen3.5 group.

	Model	MLLM4OR-Test (1,465)	LLM4OR-VL (878)	MM-OptBench (780)	Micro Avg	Macro Avg
General MLLMs	Claude Sonnet 4.6	$55.5_{\pm 6.5}$	$59.8_{\pm 5.0}$	$24.2_{\pm 4.5}$	$48.9_{\pm 5.3}$	$46.5_{\pm 5.0}$
	Gemini 3.1 Pro	$82.5_{\pm 3.2}$	$77.0_{\pm 1.5}$	$57.7_{\pm 5.2}$	$74.7_{\pm 2.6}$	$72.4_{\pm 2.7}$
	GPT-5.5	$85.3_{\pm 0.4}$	$86.4_{\pm 0.4}$	$63.4_{\pm 0.5}$	$80.1_{\pm 0.2}$	$78.3_{\pm 0.2}$
	Kimi-K2.5	$81.6_{\pm 1.3}$	$82.1_{\pm 2.8}$	$53.6_{\pm 2.8}$	$74.7_{\pm 1.7}$	$72.4_{\pm 2.0}$
	Qwen3.6-Plus	$16.3_{\pm 1.1}$	$23.1_{\pm 2.9}$	$16.3_{\pm 1.4}$	$18.2_{\pm 1.5}$	$18.6_{\pm 1.5}$
	GLM-4.5V	$37.9_{\pm 0.9}$	$46.4_{\pm 1.9}$	$11.1_{\pm 1.1}$	$33.6_{\pm 0.8}$	$31.8_{\pm 0.8}$
Math MLLMs	MathCoder-VL-8B	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$
	MM-Eureka	$15.4_{\pm 0.4}$	$33.6_{\pm 0.6}$	$0.4_{\pm 0.1}$	$16.8_{\pm 0.2}$	$16.4_{\pm 0.2}$
	MM-PRM	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$
Qwen3.5 Fine-tuned Models	Qwen3.5-9B	$26.2_{\pm 4.2}$	$28.0_{\pm 3.4}$	$11.8_{\pm 3.4}$	$23.1_{\pm 3.6}$	$22.0_{\pm 3.5}$
	Direct-SFT	$79.5_{\pm 0.4}$	$41.0_{\pm 0.9}$	$40.8_{\pm 1.3}$	$59.0_{\pm 0.6}$	$53.8_{\pm 0.7}$
	Compact-SFT	$54.1_{\pm 1.0}$	$44.8_{\pm 0.9}$	$29.7_{\pm 1.2}$	$45.4_{\pm 0.7}$	$42.9_{\pm 0.7}$
	Detailed-SFT	$43.4_{\pm 0.9}$	$29.2_{\pm 1.7}$	$29.3_{\pm 1.1}$	$35.9_{\pm 0.7}$	$34.0_{\pm 0.8}$
	Mixture-SFT	$\mathbf{91.0_{\pm 0.4}}$	$46.2_{\pm 5.7}$	$38.8_{\pm 6.5}$	$65.4_{\pm 2.5}$	$58.7_{\pm 3.0}$
	SG-DPO	$90.1_{\pm 0.8}$	$\mathbf{49.5_{\pm 2.8}}$	$\mathbf{44.3_{\pm 0.8}}$	$\mathbf{67.2_{\pm 1.0}}$	$\mathbf{61.3_{\pm 1.1}}$

A compact model can outperform commercial baselines. SG-DPO surpasses Qwen3.6-Plus and GLM-4.5V on all three suites and Claude Sonnet 4.6 in Micro/Macro accuracy (Claude's results remain provisional). Mixture-SFT and SG-DPO also outperform GPT-5.5 in-domain: Mixture-SFT reaches 91.0% versus 85.3% mean pass@1. Task-specific post-training thus makes an open 9B model competitive with several commercial systems, although GPT-5.5 retains its lead on the external suites.

Gains extend across tasks and difficulty. Mixture-SFT improves over the untuned baseline across every evaluated external source and MM-OptBench family, including routing, scheduling, and multi-period planning (§G.4; §G.3). Although their formulations differ, these tasks share the need to translate multimodal requirements into executable constraints. Gains also reach hard MM-OptBench problems (+17.0 points), consistent with reusable modeling skills rather than an improvement confined to easy cases. On MLLM4OR-Test and MM-OptBench, its mean pass@1 even exceeds the backbone's pass@4 (Table 19): additional sampling from the untuned model does not recover the trained model's single-answer accuracy, though this is not a compute-matched comparison.

Supervision aligned with optimization modeling matters. The three mathematical baselines score near zero on MM-OptBench, qualitatively agreeing with Li et al. (2026b). MM-Eureka nevertheless reaches 15.4% and 33.6% mean pass@1 on MLLM4OR-Test and LLM4OR-VL, suggesting lower effective difficulty for this model and revealing partial transfer obscured by MM-OptBench alone. Mixture-SFT and SG-DPO move beyond the near-zero regime on MM-OptBench, supporting supervision for executable optimization modeling beyond mathematical reasoning alone. This comparison concerns direct answer generation (§G.7).

4.2 RQ2: How Do Training Explanations Affect Performance?

More explanation is not better supervision. We vary explanation formats on the same training problems, requesting only formulations and code at evaluation. Compact-SFT exceeds Detailed-SFT on every suite, but not Direct-SFT consistently (Table 1). Compact annotations preserve links from text-image evidence to modeling choices while removing supporting detail. Their advantage suggests that making these links explicit can be more useful than elaborating every step, not that the task requires less reasoning.

Mixed supervision offers more than a compromise. Mixture-SFT surpasses both constituent formats on MLLM4OR-Test and LLM4OR-VL, gaining 11.4 points over the strongest single format in-domain and leading SFT Micro/Macro means. Each problem contributes one answer; neither new problems nor duplicate examples explain the gain. This suggests that direct-output practice and concise explanations provide complementary supervision. Direct-SFT remains stronger on MM-OptBench; this is a supervision-design comparison, not an equal-token ablation (§B.3).

Accuracy and coverage favor different choices. Mixture-SFT leads SFT aggregate pass@4, but Compact-SFT covers more LLM4OR-VL problems despite its lower mean pass@1 (Table 19). Thus the strongest single-answer model need not offer the widest coverage across attempts: reliable answer generation and candidate generation for subsequent verification are related but distinct goals.

4.3 RQ3: What Does DPO Add Beyond SFT?

We compare SG-DPO with its Mixture-SFT initializer under shared inference settings (§F.6).

Preference learning strengthens external transfer. SG-DPO raises mean pass@1 by 3.3 and 5.6 points on LLM4OR-VL and MM-OptBench, respectively, improving Micro/Macro means despite a small in-domain decrease (Table 1). Gains extend to seven of eight LLM4OR-VL sources and all six MM-OptBench families (§G.4; §G.3). Preference construction reuses SFT problems and verified answers, adding supervision through contrasts with the model's own failures rather than new problems. This supports preference learning as a complement to SFT for external transfer, without isolating individual DPO components.

More reliable answers and broader coverage. Higher external mean accuracy and lower run-to-run variation persist across the four sampled runs alone (§F.6). Pass@4 also improves on every suite, covering a net 36 additional problems with the same four-answer budget (Table 19). Thus, more reliable external answers accompany broader coverage across attempts. These gains concern candidate generation; selecting the correct candidate remains a separate requirement.

Successful execution does not ensure correct results. Figure 5 compares Qwen3.5-9B with all five fine-tuned models on 3,102 shared instances. Code/runtime errors dominate fine-tuned failures, while objective mismatches account for about 40% of Mixture-SFT and SG-DPO failures. Execution reliability and solution correctness therefore remain distinct challenges. Within-failure normalization means that larger shares need not imply more errors overall. Similar greedy failure totals also conceal differences in which problems Mixture-SFT and SG-DPO solve (§G.6). These descriptive profiles neither establish causal training effects nor explain five-run mean gains.

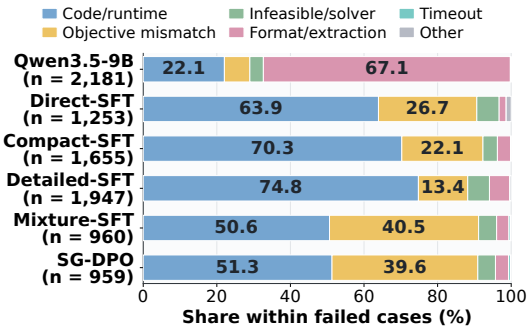


Figure 5: Greedy failure profiles on shared evaluation instances. Bars show within-model proportions; n counts failures (§G.6).

5 Conclusion

We present MLLM4OR, an open 9B model for translating complementary text-image specifications into mathematical formulations and solver code. The solver-verified MLLM4OR-Corpus supports training and in-domain evaluation; LLM4OR-VL broadens evaluation across external problem sources. Our experiments show that supervision design matters: combining direct answers with concise explanations yields the strongest aggregate SFT performance, while SG-DPO improves external accuracy and four-answer coverage, with a small in-domain trade-off. Together, these results demonstrate the potential of compact open MLLMs when trained on verified modeling examples and solver-grounded preferences. Remaining execution and objective errors motivate stronger consistency checks between the input, formulation, and implementation, moving beyond successful execution toward faithful optimization modeling.

AI USE STATEMENT

We used generative AI to assist with data reconstruction, generation and cleaning, code implementation and debugging, and language polishing. AI-assisted data and reference programs undergo the verification procedures described in §3.2 and §C–§D. The authors take responsibility for the final manuscript, artifacts, and scientific claims, including all AI-assisted material.

ETHICS STATEMENT

This study uses reconstructed benchmark problems and synthetic instances. Generated models require expert review before use.

REPRODUCIBILITY STATEMENT

Code and data: `https://anonymous.4open.science/r/ICLR-MLLM4OR-644E`. §3.2–§3.3 and §C–§D document data sources, reconstruction, filtering, deduplication, and splits. §3.4 and §E–§F specify supervision formats, training configurations, preference construction, and checkpoint selection. §B specifies task inputs and outputs, training and local inference settings, and the evaluation procedure. Archived responses, extracted programs, execution logs, scores, and provenance records support auditing and re-scoring completed runs. The completed DPO bank, training logs, and adapters are archived. §F.6 describes DPO checkpoint selection.

REFERENCES

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. *arXiv preprint arXiv:2402.10172*, 2024. URL https://arxiv.org/abs/2402.10172.

Tasnim Ahmed, Yifan Zhu, and Salimur Choudhury. Vision language models for optimization-driven intent processing in autonomous networks. *arXiv preprint arXiv:2601.12744*, 2026. URL https://arxiv.org/abs/2601.12744.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025. URL https://arxiv.org/abs/2502.13923.

Cynthia Barnhart, Peter Belobaba, and Amedeo R. Odoni. Applications of operations research in the air transport industry. *Transportation Science*, 37(4):368–391, 2003. doi: 10.1287/trsc.37.4.368.23276.

Kaushal Chari and Tarun K. Sen. An implementation of a graph-based modeling system for structured modeling (gbms/sm). *Decision Support Systems*, 22(2):103–120, 1998. doi: 10.1016/S0167-9236(97)00056-0.

Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. Solver-informed RL: Grounding large language models for authentic optimization modeling. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/98555b92e5dedbdad5921049281995bc-Abstract-Conference.html.

George B. Dantzig and John H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959. doi: 10.1287/mnsc.6.1.80.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, et al. Molmo and PixMo: Open weights and open data for state-of-the-art vision-language models. *arXiv preprint arXiv:2409.17146*, 2024. URL https://arxiv.org/abs/2409.17146.

Lingxiao Du, Fanqing Meng, Zongkai Liu, Zhixiang Zhou, Ping Luo, Qiaosheng Zhang, and Wenqi Shao. MM-PRM: Enhancing multimodal mathematical reasoning with scalable step-level supervision. *arXiv preprint arXiv:2505.13427*, 2025. URL https://arxiv.org/abs/2505.13427.

Robert Fourer, David M. Gay, and Brian W. Kernighan. A modeling language for mathematical programming. *Management Science*, 36(5):519–554, 1990. doi: 10.1287/mnsc.36.5.519.

Julia Granacher, Tuong-Van Nguyen, Rafael Castro-Amoedo, and François Maréchal. Overcoming decision paralysis—a digital twin for decision making in energy system design. *Applied Energy*, 306:117954, 2022. doi: 10.1016/j.apenergy.2021.117954.

Diwakar Gupta and Brian Denton. Appointment scheduling in health care: Challenges and opportunities. *IIE Transactions*, 40(9):800–819, 2008. doi: 10.1080/07408170802165880.

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, 2025. doi: 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:2405.13144*, 2024. URL https://arxiv.org/abs/2405.13144.

George Ioannou, Michalis N. Kritikos, and Gregory P. Prastacos. Map-route: A gis-based decision support system for intra-city vehicle routing with time windows. *Journal of the Operational Research Society*, 53(8):842–854, 2002. doi: 10.1057/palgrave.jors.2601375.

Mengzhao Jia, Zhihan Zhang, Wenhao Yu, Fangkai Jiao, and Meng Jiang. Describe-then-reason: Improving multimodal mathematical reasoning through visual comprehension training. *arXiv preprint arXiv:2404.14604*, 2024.

Zhong Li, Zihan Guo, Xiaohan Lu, Juntao Wang, Jie Song, Chao Shen, Jiageng Wu, and Mingyang Sun. OptArgus: A multi-agent system to detect hallucinations in LLM-based optimization modeling. *arXiv preprint arXiv:2605.11738*, 2026a.

Zhong Li, Qi Huang, Yuxuan Zhu, Mohammad Mohammadi Amiri, Niki van Stein, Thomas Bäck, Matthijs van Leeuwen, Zaiwen Wen, and Lincen Yang. Mm-optbench: A solver-grounded benchmark for multimodal optimization modeling. *arXiv preprint arXiv:2605.12154*, 2026b.

Zhong Li, Hongliang Lu, Tao Wei, Yuxuan Chen, Wenyu Liu, Yuan Lan, Fan Zhang, and Zaiwen Wen. Constructing industrial-scale optimization modeling benchmark. In *Forty-third International Conference on Machine Learning*, 2026c. URL https://openreview.net/forum?id=0Q7adIsv6p.

Yufang Liu, Yao Du, Tao Ji, Jianing Wang, Yang Liu, Yuanbin Wu, Aimin Zhou, Mengdi Zhang, and Xunliang Cai. The role of visual modality in multimodal mathematical reasoning: Challenges and insights. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22596–22611, 2025.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. Optmath: A scalable bidirectional data synthesis framework for optimization modeling. In *International Conference on Machine Learning*, pp. 40769–40802. PMLR, 2025.

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.

Pan Lu, Ran Gong, Shibiao Jiang, Liang Qiu, Siyuan Huang, Xiaodan Liang, and Song-Chun Zhu. Inter-GPS: Interpretable geometry problem solving with formal language and symbolic reasoning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 6774–6786, 2021.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=KUNzEQMWU7.

Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952. doi: 10.1111/j.1540-6261.1952.tb01525.x.

Ahmed Masry, Xuan Long Do, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. Chartqa: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the association for computational linguistics: ACL 2022*, pp. 2263–2279, 2022.

Fanqing Meng, Lingxiao Du, Zongkai Liu, Zhixiang Zhou, Quanfeng Lu, Tiancheng Han, Daocheng Fu, Botian Shi, Wenhai Wang, Junjun He, Kaipeng Zhang, Ping Luo, Yu Qiao, Qiaosheng Zhang, and Wenqi Shao. MM-Eureka: Exploring the frontiers of multimodal reasoning with rule-based reinforcement learning. *arXiv preprint arXiv:2503.07365*, 2025. URL https://arxiv.org/abs/2503.07365.

Narayana Prasad Padhy. Unit commitment—a bibliographical survey. *IEEE Transactions on Power Systems*, 19(2):1196–1205, 2004. doi: 10.1109/TPWRS.2003.821611.

Hasan Pirkul, Rakesh Gupta, and Erik Rolland. Visopt: A visual interactive optimization tool for p-median problems. *Decision Support Systems*, 26(3):209–223, 1999. doi: 10.1016/S0167-9236(99)00032-9.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html.

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, pp. 189–203. PMLR, 2023. URL https://proceedings.mlr.press/v220/ramamonjison23a.html.

Luıs Santos, João Coutinho-Rodrigues, and Carlos Henggeler Antunes. A web spatial decision support system for vehicle routing using google maps. *Decision Support Systems*, 51(1):1–9, 2011. doi: 10.1016/j.dss.2010.11.008.

Nilay Shah. Process industry supply chains: Advances and challenges. *Computers & Chemical Engineering*, 29(6):1225–1235, 2005. doi: 10.1016/j.compchemeng.2005.02.023.

Wenhao Shi, Zhiqiang Hu, Yi Bin, Junhua Liu, Yang Yang, See Kiong Ng, Lidong Bing, and Roy Ka-Wei Lee. Math-llava: Bootstrapping mathematical reasoning for multimodal large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 4663–4680, 2024.

Dong Sun, Renfei Huang, Yuanzhe Chen, Yong Wang, Jia Zeng, Mingxuan Yuan, Ting-Chuen Pong, and Huamin Qu. Planningvis: A visual analytics approach to production planning in smart factories. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):579–589, 2020. doi: 10.1109/TVCG.2019.2934275.

Ke Wang, Junting Pan, Linda Wei, Aojun Zhou, Weikang Shi, Zimu Lu, Han Xiao, Yunqiao Yang, Houxing Ren, Mingjie Zhan, and Hongsheng Li. MathCoder-VL: Bridging vision and code for enhanced multimodal mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 2505–2534, 2025. doi: 10.18653/v1/2025.findings-acl.128. URL https://aclanthology.org/2025.findings-acl.128/.

Zhuohan Wang, Ziwei Zhu, Ziniu Li, Congliang Chen, Zhihang Lin, Mingzhe Yang, Yizhou Han, Yufeng Lin, Angyang Gu, Xinglin Hu, Ruoyu Sun, and Tian Ding. Certified evaluation for LLMs in optimization modeling: From graph isomorphism to formulation isomorphism. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, 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, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Thomas R. Willemain. Model formulation: What experts think about and when. *Operations Research*, 43(6):916–932, 1995. doi: 10.1287/opre.43.6.916.

Pak Chung Wong, Zhenyu Huang, Yousu Chen, Patrick S. Mackey, and Shuangshuang Jin. Visual analytics for power grid contingency analysis. *IEEE Computer Graphics and Applications*, 34(1): 42–51, 2014. doi: 10.1109/MCG.2014.17.

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

Ziyang Xiao, Jingrong Xie, Lilin Xu, Shisi Guan, Jingyan Zhu, Xiongwei Han, Xiaojin Fu, WingYin Yu, Han Wu, Wei Shi, et al. A survey of optimization modeling meets LLMs: Progress and future directions. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pp. 10742–10750, 2025.

Wantong Xie, Yi-Xiang Hu, Jieyang Xu, Feng Wu, and Xiangyang Li. MURKA: Multi-reward reinforcement learning with knowledge alignment for optimization tasks. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://openreview.net/forum?id=f4pvPNf9ox.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. OptiBench meets ReSocratic: Measure and improve LLMs for optimization modeling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=fsDZwS49uY.

Renrui Zhang, Dongzhi Jiang, Yichi Zhang, Haokun Lin, Ziyu Guo, Pengshuo Qiu, Aojun Zhou, Pan Lu, Kai-Wei Chang, Yu Qiao, et al. Mathverse: Does your multi-modal llm truly see the diagrams in visual math problems? In *European Conference on Computer Vision*, pp. 169–186. Springer, 2024a.

Renrui Zhang, Xinyu Wei, Dongzhi Jiang, Ziyu Guo, Yichi Zhang, Chengzhuo Tong, Jiaming Liu, Aojun Zhou, Shanghang Zhang, Peng Gao, and Hongsheng Li. MAVIS: Mathematical visual instruction tuning with an automatic data engine. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=MnJzJ2gvuf.

Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. Multimodal chain-of-thought reasoning in language models. *Transactions on Machine Learning Research*, 2024b. URL https://arxiv.org/abs/2302.00923.

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*, 2026. URL https://openreview.net/forum?id=ZrgxU8WMmG.

Post-Training Open Multimodal Models for Optimization Modeling from Text and Images

APPENDIX CONTENTS

A Extended Related Work . . . 14
A.1 Automated Optimization Modeling . . . 15
A.2 Visual Evidence and Multimodal Mathematical Reasoning . . . 15
A.3 SFT and Alignment for Multimodal Mathematics . . . 16
A.4 Outcome, Process, and Solver Feedback . . . 16

B Experimental Setup and Evaluation Protocol . . . 17
B.1 Model Inputs and Output Requirements . . . 17
B.2 Code Execution and Scoring . . . 17
B.3 Training and Local Inference Settings . . . 18
B.4 Evaluation Metrics and Reporting . . . 18

C MLLM4OR-Corpus: Construction, Verification, and Statistics . . . 19
C.1 Construction and Quality Control . . . 19
C.2 Multimodal Reconstruction from OptMATH-Train . . . 20
C.3 Multimodal Reconstruction from OR-Instruct-3K . . . 21
C.4 Parametric Generation of Multimodal Problems . . . 22
C.5 Automated Checks and Reference Code Verification . . . 23
C.6 Deduplication and Dataset Splits . . . 25
C.7 Final Corpus Composition . . . 26
C.8 Corpus Statistics and Reference-Model Sizes . . . 26
C.9 Corpus Scope and Limitations . . . 28

D Evaluation Datasets . . . 28
D.1 Test Suites and Development Set . . . 28
D.2 MLLM4OR-Test and MM-OptBench . . . 28
D.3 LLM4OR-VL Sources and Selection . . . 29
D.4 Problem Reconstruction and Text–Image Design . . . 32
D.5 Quality Control and Dataset Overlap . . . 32
D.6 Multimodal and Text-Only Inputs . . . 33
D.7 LLM4OR-VL Input Statistics . . . 33

E SFT Data and Supervision . . . 34
E.1 SFT Reference Answers . . . 34
E.2 Mixture-SFT Construction . . . 35

F Solver-Grounded Direct Preference Optimization . . . 36
F.1 Rationale and Training Procedure . . . 36
F.2 Response Verification and Pair Eligibility . . . 37
F.3 Generating Candidate Responses . . . 38
F.4 Constructing Preference Pairs . . . 38
F.5 DPO Training Configuration . . . 38
F.6 Evaluation Protocol and Scope . . . 38

G Supplementary Results and Analysis . . . 39
G.1 Four-Answer Coverage (Pass@4) . . . 40
G.2 Single-Format SFT Results . . . 40
G.3 MM-OptBench Results by Family and Difficulty . . . 40
G.4 LLM4OR-VL Results by Source . . . 42
G.5 MLLM4OR-Test Results by Source . . . 42
G.6 Failure Types and Paired Changes . . . 43
G.7 Mathematical-Baseline Evaluation Scope . . . 44

A EXTENDED RELATED WORK

We review four strands of work relevant to MLLM4OR: automated optimization modeling (§A.1), multimodal mathematical reasoning (§A.2), explanation supervision and alignment (§A.3), and feedback-based learning (§A.4). The comparison highlights how prior work informs our task setting, training targets, and use of solver feedback.

A.1 Automated Optimization Modeling

Natural-language-to-optimization (NL-to-Opt) research studies how to translate textual problem descriptions into mathematical formulations and executable solver programs. Early benchmarks such as NL4Opt focus on identifying and representing linear-program components (Ramamonjison et al., 2023). ComplexOR (Xiao et al., 2024), IndustryOR (Huang et al., 2025), MAMO (Huang et al., 2024), OptiBench (Yang et al., 2025), and OptMATH-Bench (Lu et al., 2025) broaden the range of problem structures and application domains. Recent benchmarks address two further challenges: Bench4Opt evaluates formulation-level correctness (Wang et al., 2026), while MIPLIB-NL preserves the structure and scale of industrial optimization problems (Li et al., 2026c). Together, these efforts emphasize the need to assess both modeling fidelity and performance on practically relevant instances.

Methods in this area improve either the inference workflow or the model itself. Chain-of-Experts (Xiao et al., 2024) and OptiMUS (AhmadiTeshnizi et al., 2024) divide modeling into multiple calls and use feedback to revise generated programs. ORLM instead trains models through optimization-specific instruction tuning (Huang et al., 2025). ReSocratic constructs structured modeling demonstrations before generating corresponding problem descriptions (Yang et al., 2025), while OptMATH combines bidirectional data synthesis with rejection sampling (Lu et al., 2025). These approaches motivate task-specific training data; solver-based learning adds feedback on generated solutions (§A.4).

IntentOpt (Ahmed et al., 2026) and MM-OptBench (Li et al., 2026b) extend optimization modeling to multimodal inputs and are the closest task settings to ours. IntentOpt evaluates solver-code generation from network diagrams and textual requirements. MM-OptBench covers six optimization families and requires both a mathematical formulation and solver code. Both focus on evaluating existing models. Our work complements these evaluations with a solver-verified corpus for training, model selection, and evaluation, an additional reconstructed test suite, and post-training methods for learning from complementary text–image inputs.

A.2 Visual Evidence and Multimodal Mathematical Reasoning

Table 2: Representative tasks related to multimodal optimization modeling, compared by input evidence and required outputs.

Representative work	Input evidence	Required reasoning or output
ChartQA (Masry et al., 2022)	Chart images and questions.	Answers obtained by reading and reasoning over plotted values.
Inter-GPS (Lu et al., 2021)	Geometry text and diagrams.	Formal geometric relations and symbolic reasoning to answer the question.
MathVista / MathVerse (Lu et al., 2024; Zhang et al., 2024a)	Mathematical questions with varied visual evidence and textual detail.	Answers to mathematical questions; MathVerse varies access to information across modalities.
IntentOpt (Ahmed et al., 2026)	Network diagrams and textual requirements; 85 instances.	Executable network-optimization code, evaluated through solver execution.
MM-OptBench (Li et al., 2026b)	Text–image specifications across six optimization families; 780 instances.	A mathematical formulation and executable solver code for each decision problem.
MLLM4OR-Corpus / LLM4OR-VL (ours)	Complementary text–image inputs from reconstructed and generated optimization problems.	The same formulation-and-code task, with a corpus for post-training and evaluation plus an additional test suite.

Visual mathematical reasoning requires models to extract evidence from images and connect it to a formal solution. ChartQA studies numerical reasoning over chart images (Masry et al., 2022), while Inter-GPS translates geometry text and diagrams into formal relations for symbolic reasoning (Lu et al., 2021). These tasks establish visual interpretation and formal reasoning as complementary components of problem solving.

MathVista evaluates mathematical reasoning across diverse visual contexts (Lu et al., 2024). MathVerse varies the information available in text and images (Zhang et al., 2024a), and HC-M3D tests image reliance through visually similar problems with different answers (Liu et al., 2025). These benchmarks are relevant to our use of complementary modalities, where the text alone does not

fully specify the problem. The required output is also different: multimodal optimization modeling translates operational requirements and visual data into decision variables, an objective, constraints, and executable solver code. Table 2 summarizes these differences in inputs and outputs.

A.3 SFT AND ALIGNMENT FOR MULTIMODAL MATHEMATICS

Table 3: Representative post-training designs and their relation to our explanation supervision and solver-grounded preferences.

Representative work	Main design choice	Comparison with this study
Multimodal-CoT / VCAR (Zhang et al., 2024b; Jia et al., 2024)	Separate rationale or visual-description generation from subsequent reasoning.	We vary explanations during training and request only a formulation and solver code at evaluation.
Math-LLaVA (Shi et al., 2024)	Combine curated examples with synthesized questions to expand training coverage.	We keep the training problems fixed and vary the explanations in their reference answers.
MathCoder-VL (Wang et al., 2025)	Align images with rendering code before mathematical instruction tuning.	Our code implements an optimization problem rather than rendering a figure.
MAVIS (Zhang et al., 2025)	Follow visual alignment and mathematical instruction tuning with DPO.	We construct preferences from solver-checked failures on text–image optimization problems.

Reasoning explanations can be used as training targets or generated as intermediate steps at inference. CoT prompting elicits intermediate reasoning (Wei et al., 2022); Multimodal-CoT separates visually grounded rationale generation from answer inference (Zhang et al., 2024b). Describe-then-Reason (VCAR) similarly separates visual description from subsequent reasoning using two LoRA adapters (Jia et al., 2024). These methods make intermediate reasoning part of the inference procedure. We instead study explanations as training supervision: all SFT variants receive the same instruction to produce only a formulation and solver code at evaluation.

Other approaches improve the training data or the representation of visual information. Math-LLaVA combines curated examples with synthesized questions to expand training coverage (Shi et al., 2024). MathCoder-VL uses image-to-rendering-code supervision before mathematical instruction tuning (Wang et al., 2025). Its rendering code describes how to draw a figure, whereas our solver code implements the decision problem specified by the text and image. Rather than varying the problem set or output representation, our SFT comparison fixes the problems, reference formulations, and solver programs, and varies the accompanying explanations. Table 3 places these choices alongside representative post-training methods.

The amount of explanation is a separate design choice. On MathVista, Jia et al. (2024) report 33.1% accuracy for VCAR with GPT-4V supervision and 33.7% with Gemini-Pro supervision (their Table 3). They suggest that longer, more complex responses may be harder for LLaVA-7B to learn from. Because the teacher also differs, this comparison does not isolate explanation length. It nevertheless motivates examining the amount of reasoning supervision rather than assuming that more detail is always beneficial. We study both explanation detail and the proportion of training answers containing explanations (§3.4.1), while recognizing that the same number of training problems does not imply the same answer-token budget.

A.4 OUTCOME, PROCESS, AND SOLVER FEEDBACK

Feedback can assess final answers, intermediate reasoning, or executable programs, and can guide either training or inference. MM-Eureka uses outcome-based reinforcement learning for multimodal mathematics (Meng et al., 2025). MM-PRM learns to score reasoning steps from tree-search-generated labels and ranks candidate solutions at inference (Du et al., 2025). MAVIS follows mathematical instruction tuning with DPO (Zhang et al., 2025). These approaches establish that combining SFT with feedback-based learning is not itself new; the key choices concern what is verified and how the feedback is used.

For optimization modeling, solvers provide feedback on generated formulations and programs. Solver-Informed RL uses execution and instance-level model checks to reward syntactic validity, feasibility, and solution quality (Chen et al., 2025). MURKA combines multi-reward reinforcement learning and distillation across extraction, solving, and checking agents (Xie et al., 2025). StepORLM combines solver-based outcome verification with a learned generative process reward model and updates the policy through weighted DPO (Zhou et al., 2026). PEARL instead learns when to execute partial models and revise them during an interactive modeling trajectory (Lu et al., 2026). Solver feedback thus supports several distinct uses: policy training, process evaluation, and inference-time revision.

Our approach uses the standard DPO objective (Rafailov et al., 2023) with a fixed set of preferences constructed before training. Each pair contrasts a verified reference answer with the SFT model's own failure on the same text–image input. These failures include programs that execute successfully but return an incorrect objective value. Unlike StepORLM's learned process verifier or PEARL's interactive policy, our method requires neither a learned reward model nor solver calls during policy updates; evaluation also remains single-call. The contribution is the construction of preferences for multimodal formulation-and-code generation (§3.4.2), not a new preference objective. Execution feedback does not, however, certify semantic equivalence; OptArgus addresses the complementary task of checking consistency among descriptions, formulations, and programs (Li et al., 2026a).

B Experimental Setup and Evaluation Protocol

This appendix specifies the model inputs and required outputs (§B.1), code execution and scoring (§B.2), training and local inference settings (§B.3), and evaluation metrics (§B.4).

B.1 Model Inputs and Output Requirements

Each evaluation input consists of a problem statement and one or more images. The text describes the decision task, objective, and operational rules; the images provide numerical data or relational structure. Models receive neither private reference answers nor reference programs or execution outcomes.

The output instruction requests a formulation in `<mathematical_model>` followed by self-contained Python code in `<solver_code>`. The program must derive the instance data from the inputs, solve the problem with Gurobi through `gurobipy`, and print a final JSON record prefixed by `RESULT_JSON:`. This record contains `status` and `objective_value`; feasibility tasks also require an assignment when feasible, while multiple scenarios use `objective_values` as an ordered list or scenario-keyed object. Hard-coded reference optima are prohibited.

Evaluation uses one model call per response, without execution feedback or iterative repair. All trained variants, including those trained with explanations, are asked to return only the formulation and code. Primary scores use generated programs without modifying their statements; the actual acceptance checks are specified below.

B.2 Code Execution and Scoring

We check Python syntax, permitted operations, and use of Gurobi before executing each unmodified program in a separate process with a 600-second limit. By default, we compare the solver-computed objective value $\hat{z}$ with the private reference optimum $z^\star$ for the same instance. The program must finish without errors, and the finite objective values must satisfy

$$|\hat{z} - z^\star| \le \max\left\{\epsilon_{\text{abs}},\ \epsilon_{\text{rel}} \max\left(|\hat{z}|, |z^\star|\right)\right\}.$$

The default absolute and relative tolerances are $\epsilon_{\text{abs}} = 10^{-6}$ and $\epsilon_{\text{rel}} = 10^{-5}$, respectively. The absolute tolerance handles values near zero, while the relative tolerance scales with objective magnitude. Private references are used only for scoring and are not exposed to the program.

Feasibility tasks require a valid assignment when feasible, or the appropriate infeasibility status otherwise. Passing these checks does not prove formulation equivalence.

B.3 Training and Local Inference Settings

Model selection uses all 1,453 MLLM4OR-Dev instances: SFT selects checkpoints and mixture ratios, and DPO selects among its saved checkpoints (§F.6). Development scores are not included in test averages. None of the three test suites is used for training or building preference pairs.

Our local experiments use one NVIDIA H200 GPU and multiple NVIDIA H100 GPUs. Each SFT run uses four H100 GPUs, and H100 GPUs also support preference rollouts, checkpoint-evaluation inference, and final test inference. The reported DPO optimization run uses the H200.

The reported untuned Qwen3.5-9B baseline uses an API, whereas the post-trained models use local inference. Their comparison is therefore not a matched-deployment ablation.

Table 4 lists the SFT training settings. All variants train on the same 19,189 problems. However, their answers can differ in length, and their selected checkpoints can come from different training steps. Their training token counts and computational costs can therefore differ. §E.1 explains how we prepare reference answers, and §E.2 explains how we assign answer formats in Mixture-SFT.

Table 4: SFT configuration. The sequence limit includes both the input and reference answer.

Shared setting	**Value (all SFT variants)**
Backbone	Qwen3.5-9B
Training problems	Same 19,189 instances from MLLM4OR-Train
LoRA configuration	All-linear; rank 32, $\alpha = 64$, dropout 0.05; aligner frozen
Training loss	Next-token prediction on reference answers only
Learning rate	10^{-4}; cosine decay; 5% warm-up
Training budget	4,798 optimizer updates (one nominal epoch)
Batch / hardware	Effective batch four; one example per GPU; four NVIDIA H100 GPUs per SFT run
Other settings	bfloat16; weight decay 0.1; gradient checkpointing; SDPA
Random seeds	Training seed 42; data-shuffling seed 42
SFT variant	**Training sequence limit**
Direct-SFT, Compact-SFT, Mixture-SFT	32,768 tokens
Detailed-SFT	49,152 tokens

At evaluation, the SFT models can generate up to 32,768 tokens per answer, or 49,152 for Detailed-SFT. Each test instance receives one greedy answer and four sampled answers. The SFT and DPO evaluations use the same sampling temperature and top-p across all three test suites. The four additional answers use random seeds 101, 202, 303, and 404. The image-token limit is 16,384. Detailed-SFT's parent training-run identifier is missing from the available summary, although its checkpoint number is recorded. DPO evaluation settings are given in §F.6.

To build DPO preference pairs, we sample two answers per problem from Mixture-SFT at temperature 0.7 and top-$p = 0.95$, with up to 16,384 tokens per answer. §F describes how we form the pairs and train with DPO.

B.4 Evaluation Metrics and Reporting

Pass@1. For a fixed model, let $b_{ir} \in \{0, 1\}$ indicate whether the scored response to instance i passes in run r. Run $r = 0$ is the first response and $r = 1, \ldots, 4$ are the four additional responses. For five runs covering the same complete set of N instances, we compute run accuracy a_r, mean $\bar{a}$, and sample standard deviation (SD) s:

$$a_r = \frac{100}{N} \sum_{i=1}^{N} b_{ir}, \qquad \bar{a} = \frac{1}{5} \sum_{r=0}^{4} a_r, \qquad s = \sqrt{\frac{1}{4} \sum_{r=0}^{4} (a_r - \bar{a})^2}. \tag{2}$$

We report pass@1 as $\bar{a}_{\pm s}$, following MM-OptBench's five-run convention (Li et al., 2026b). Accuracy is expressed in percent and SD in percentage points. The SD describes accuracy variation over one

greedy and four sampled runs; it is not a confidence interval or variability across independently trained models.

Pass@4. Pass@4 is $100N^{-1}\sum_{i=1}^{N}\max_{1\le r\le 4} b_{ir}$, the percentage of instances solved by at least one of four additional answers. It requires four matched, scored answers per instance and excludes the first answer. It measures candidate coverage, not automatic answer selection; a single four-answer union has no run-level SD.

Micro and macro averages. Let $a_{\mathrm{T},r}$, $a_{\mathrm{VL},r}$, and $a_{\mathrm{MM},r}$ denote run-level percentage accuracies on MLLM4OR-Test, LLM4OR-VL, and MM-OptBench. Micro averages weight suites by size, whereas macro averages weight them equally:

$$\begin{aligned} a_{\mathrm{micro},r} &= \frac{1465a_{\mathrm{T},r} + 878a_{\mathrm{VL},r} + 780a_{\mathrm{MM},r}}{3123}, \\ a_{\mathrm{macro},r} &= \frac{a_{\mathrm{T},r} + a_{\mathrm{VL},r} + a_{\mathrm{MM},r}}{3}. \end{aligned} \tag{3}$$

We aggregate corresponding runs before computing their mean and SD, rather than averaging suite-level SDs. Pass@4 uses the same suite weights. LLM4OR-VL pools its 878 instances; its eight sources are not additional test suites. Development data are excluded, and all statistics use exact counts before rounding.

C MLLM4OR-Corpus: Construction, Verification, and Statistics

This appendix explains how we construct and verify MLLM4OR-Corpus. We first describe the shared construction and review process (§C.1), then explain reconstruction from OptMATH-Train and OR-Instruct-3K and generation of new problems (§C.2–§C.4). We then describe automated checks, dataset splits, final counts, and corpus statistics (§C.5–§C.8), and conclude with the scope and limitations of the corpus (§C.9).

C.1 Construction and Quality Control

Each corpus instance represents one optimization problem. It contains a text–image input, a reference formulation, solver code, and a private verified result, together with the source information. All four SFT variants use the same training problems; different explanations of the same problem are not counted as separate instances.

Preparing the problem specification. For OptMATH and OR-Instruct, we start from the original text question; for generated problems, we start from the sampled configuration. We identify the entities, numerical data and units, decision variables and their domains, objective, and constraints. Any supplied solution or code is checked against these requirements. We may correct wording or translation without changing the problem. Missing data, conflicting units, unclear variable domains, or unspecified objective priorities must be resolved from the source; otherwise, the instance is excluded. We keep the original source or generation configuration alongside the reconstruction records. The generator sampling procedure is described in §C.4.

Dividing information between text and image. The image shows selected numerical data or relationships, while the text states the decision task and the remaining rules. Together, they must provide enough information to formulate the problem, with consistent labels and units. For example, a network diagram shows arc data, not an optimal route. Dense networks can be shown as matrices when this makes their values easier to read. Neither modality reveals the reference formulation, solver code, optimal objective value, or optimal decisions.

Rendering the images. We render images directly from structured problem data, preserving exact values, labels, and relationships. Numerical content is not produced by a text-to-image model. Layout changes improve readability without altering the underlying problem.

Human review. Our six reviewers include graduate students, postdoctoral researchers, and professors with expertise in LLMs, machine learning, or operations research. Two reviewers examine each retained instance using the criteria in Table 5. They compare the source question or sampled configuration, text–image input, reference formulation and code, and automated check results. MLLMs assist with drafting and revision; human reviewers decide whether the content is correct. A third reviewer resolves ambiguous cases or substantive disagreements.

Automated checks. Table 5 summarizes the six quality checks in Figure 1. Automated checks complement human review by detecting missing data, execution failures, possible duplicates, and layout problems. Their implementations vary by source; §C.5 describes the reference-code checks.

Table 5: Quality checks used in corpus construction (Figure 1). Automated checks vary by source. Gurobi execution checks use standardized reference programs, which may differ from the original source code (§C.5).

Check	Automated checks	Human review
Source consistency	Source/configuration links, required fields, and file hashes.	Compare entities, data, units, variable domains, objectives, and constraints with the source or sampled configuration. Reject unsupported interpretations.
Visual review	Image links and dimensions, nonblank output, font and boundary warnings, and repeatable rendering.	Check values, labels, arrows, axes, and legends. Confirm that text and image jointly specify the problem and can be read correctly.
Model–code match	Required reference fields and Python syntax.	Trace variables, coefficients, objectives, and constraints from the problem specification to the formulation and code. Check for missing or unintended restrictions.
Gurobi execution	Execution status, outputs, and agreement with the verified reference result.	Investigate failures and mismatches against the intended problem. Do not change the problem to fit a solver result.
Leakage screening	Allowed input fields and text-pattern checks for exposed answers, code, or review notes.	Inspect both modalities for optimal values, selected decisions, solution-revealing highlights, or hints about the reference formulation.
Deduplication	Problem and complete-input fingerprints, image comparisons, and candidate template groups.	Distinguish shared templates from duplicate problems and resolve conflicting references. Do not count confirmed duplicates as new instances.

Corrections and rechecking. Each instance is accepted, corrected and checked again, or excluded. Presentation changes require renewed readability, text–image consistency, and leakage checks. Changes to problem data, formulations, or code also require semantic review and another reference execution. Unresolved content errors or missing required execution evidence prevent inclusion. A timeout means verification is incomplete, not that the problem is invalid.

Estimated review workload. Two reviews of each of the 22,107 retained instances give 44,214 assessments. At the authors' estimated one minute per assessment, this amounts to approximately 740 person-hours. This estimate is not a time log and excludes construction, rejected candidates, corrections, re-review, adjudication, and model or solver runtime.

C.2 Multimodal Reconstruction from OptMATH-Train

Source and candidate selection. We use a fixed copy of OptMATH-Train containing 201,071 text-only problem records (Lu et al., 2025). Each record includes instructions, a problem statement, a proposed formulation or program, and an answer. We reconstruct the problem from its statement and verify the supplied solution before using it as a reference. Deterministic sampling favors clear descriptions, data suitable for visual presentation, and numerically checkable answers. It uses family quotas, flexible limits on difficulty and layout categories, and a cap of 30 candidates per coarse group

defined by source metadata and basic text features. These groups guide sampling; duplicate checks are described in §C.6.

Text–image reconstruction. We extract the entities, numerical data, and operational rules, removing modeling hints, solution fragments, and coding instructions. Reviewers check completeness, units, variable domains, objectives, and constraints; corrections must be supported by the source. The checked data are rendered as matrices, networks, schedules, or other labeled displays. For a flow problem, the image shows arc directions, costs, and capacities, while the text states the decision task and remaining rules. Dense networks can use a matrix instead. We prepare a reference formulation and solver program, then verify the result through the review process in §C.1. Reconstruction changes the presentation of the source problem rather than generating new numerical data.

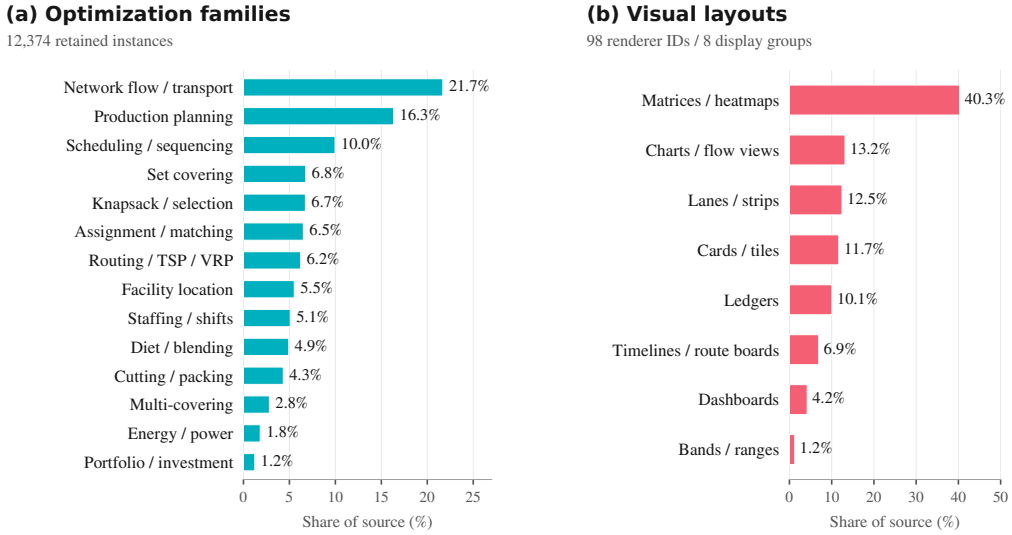


Figure 6: The 12,374 OptMATH-derived corpus instances by (a) reviewed optimization family and (b) visual format, grouping 98 rendering layouts into eight categories. Percentages include all training, development, and test instances.

Final retained instances. The final OptMATH-derived set contains 12,374 instances: 9,877 for training, 1,243 for development, and 1,254 for testing (Table 7).

Reading the source profile. Figure 6 shows the reviewed problem families and visual formats of the retained instances. The 98 rendering layouts are grouped into eight visual formats for display. §C.8 describes the corpus-wide statistics.

C.3 Multimodal Reconstruction from OR-Instruct-3K

Source and full review. We use all 3,000 records in a fixed copy of OR-Instruct (Huang et al., 2025). This English-language collection pairs industry-inspired optimization questions with proposed formulations and solver code, but has no separate answer field. Unlike the candidate sampling used for OptMATH, we review every record. We check whether the question defines a complete, consistent problem and whether the supplied solution matches it; the supplied solution is not treated as a verified reference.

Text–image reconstruction. We remove generic solver-code instructions and retain the original question's entities, numerical data, and business rules. Corrections must be supported by the source; missing costs, capacities, variable domains, or objective priorities cannot be invented to retain an instance. The checked data are rendered as matrices, tables, timelines, or other labeled displays. For example, assignment costs appear in labeled matrix cells, while resource coefficients can be shown as bars with exact values. The accompanying text states the decision task and remaining business rules.

Verification and visual review. We construct a reference formulation and solver program and check that both represent the problem's requirements. Program execution and review of the resulting objective and solution establish the reference answer. Reviewers also check that the text–image input is complete, readable, and free of solution hints, following §C.1 and §C.5.

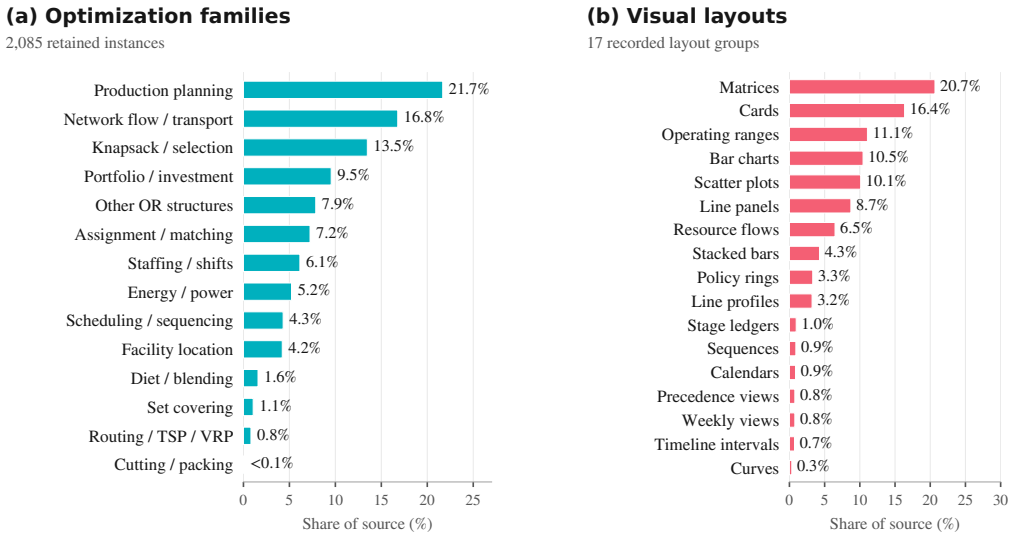


Figure 7: The 2,085 OR-Instruct-derived corpus instances by (a) reviewed optimization family and (b) visual layout. Percentages include all training, development, and test instances; all 17 layout categories are shown.

Final retained instances. The final OR-Instruct-derived set contains 2,085 instances: 1,664 for training, 210 for development, and 211 for testing (Table 7).

Reading the source profile. Figure 7 shows the 14 reviewed problem families and 17 visual-layout categories. Layouts describe image formats, such as matrices or timelines. These source-specific categories differ from OptMATH's grouped visual formats.

C.4 Parametric Generation of Multimodal Problems

Generation and sampling. Our parametric generators produce 7,648 training instances across six optimization families and 26 subfamilies. Their task families and visual designs are inspired by MM-OptBench (Li et al., 2026b). Each generator defines a problem template and varies its structure or numerical parameters under easy, medium, and hard settings. We retain the sampled configurations and random seeds.

Difficulty settings. Difficulty is specified through problem size and constraints, not measured from model performance. For example, the energy-dispatch/unit-commitment generator uses 3–4 generating units over 4–6 periods for easy instances, 4–6 units over 10–16 periods for medium instances, and 6–10 units over 18–30 periods for hard instances. Only the medium and hard settings include ramping and minimum up/down-time constraints. Demand patterns, capacity limits, and initial operating states also vary within each setting. These labels describe settings within a subfamily, not a common difficulty scale across different tasks.

Text–image generation. Each sampled configuration supplies the text, image, reference formulation, and solver program. For energy dispatch, it records the generating units, periods, demand, costs, capacities, and operating limits. The image shows unit costs, labeled demand curves, and output ranges; the text states the dispatch objective and balance, transition, and cost rules. Executing the reference program provides the private answer. Other subfamilies use networks, matrices, schedules,

maps, or resource tables. Layout and wording variants change the presentation without changing the problem, and dense data can be shown in tables rather than crowded diagrams. Unlike the two reconstructed sources, this process starts from a parameterized configuration rather than an existing text question. Figure 9 shows six examples.

Verification and visual review. Acceptance requires a finite optimum, agreement between the stored answer and reference execution, and complete, readable text–image inputs without solution hints. When several solutions are equally optimal, explicit tie-breaking rules may select a fixed reference solution; this does not make the original optimum unique. Semantic and execution checks follow §C.1 and §C.5.

Final retained instances. All 7,648 retained instances belong to MLLM4OR-Train and are used in every SFT variant; this branch contributes no instances to MLLM4OR-Dev or MLLM4OR-Test. The final set contains 2,584 easy, 2,583 medium, and 2,481 hard instances across 41 layout categories. Table 6 gives the family counts.

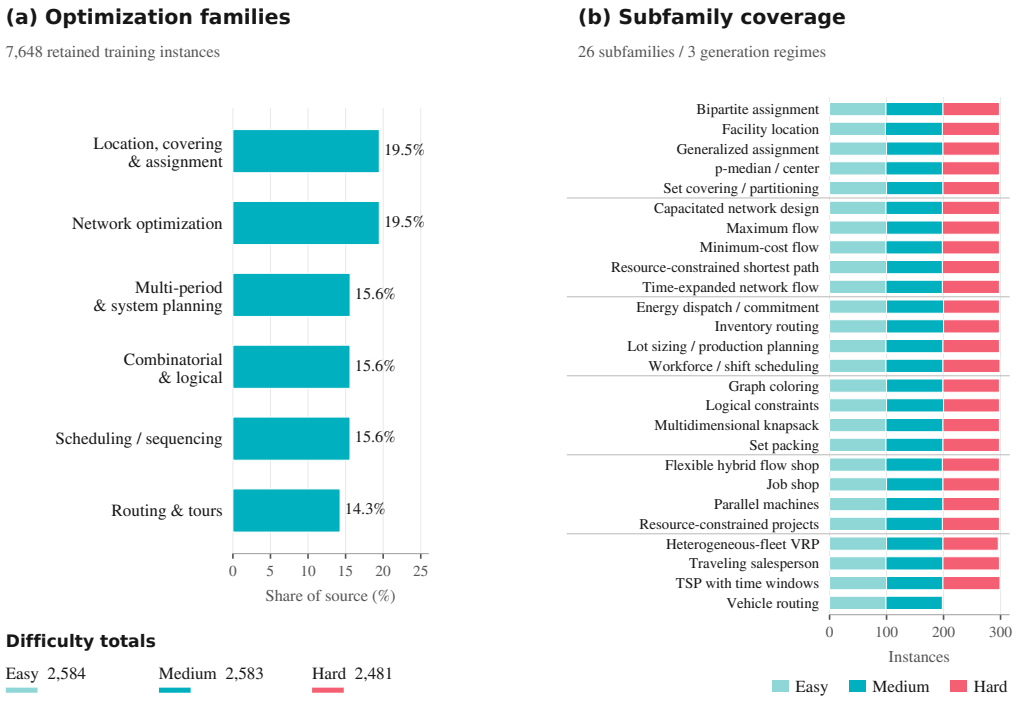


Figure 8: The 7,648 generated training instances: (a) shares of the six problem families and (b) counts by subfamily and difficulty setting.

Reading the source profile. Figure 8 shows family shares and subfamily counts by difficulty setting. Empty segments, including hard vehicle routing, indicate that no retained training instances belong to that group. Difficulty labels describe generation settings, not measured model performance.

C.5 Automated Checks and Reference Code Verification

Automated checks support the expert review described in §C.1. They identify missing data, mismatched files, possible answer leakage, and program errors. The checks vary by source, as described in §C.2–§C.4. After a correction, we repeat the affected checks before accepting the instance.

Input consistency and answer leakage. We check that records contain the required fields and the correct images. Automated layout checks, where supported, flag overlapping or out-of-bounds labels. Text scans flag exposed answers, solver code, or review notes. Reviewers inspect both modalities

(a) Assignment

Pairing cost | - = not permitted

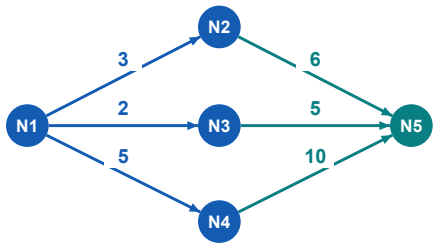
	R1	R2	R3	R4	R5	R6
D1	2	-	7	-	-	-
D2	-	3	3	-	-	-
D3	-	-	-	4	-	6
D4	-	6	-	-	5	-
D5	-	-	6	-	-	5
D6	-	9	-	6	-	3

Task: Pair every driver with one route and every route with one driver. Minimize total pairing cost.

mmobgen_ba_easy_000001

(b) Network flow

Directed arcs | labels = capacity



Task: Send as much flow as possible from source N1 to sink N5. Respect arc capacities and conserve intermediate flow.

mmobgen_maxflow_easy_000001

(c) Energy planning

Period demand



Unit	Unit cost	Startup	Min	Max
G1	12	67	5	32
G2	16	12	1	15
G3	7	69	14	36

Task: Meet demand exactly at minimum generation and startup cost. Choose on/off states and output; all units start off.

mmobgen_educ_easy_000001

(d) Production scheduling



Processing time

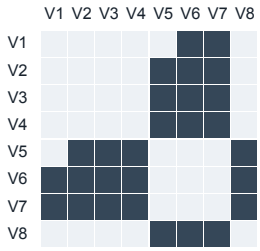
Job	S1	S2	S3
J1	7	4	2
J2	7	7	2
J3	9	9	4
J4	8	5	7
J5	8	5	3

Task: Process each job through S1, S2, then S3, without preemption or machine overlap. Minimize the final completion time.

mmobgen_fhfss_easy_000001

(e) Graph coloring

Adjacency matrix | filled cell = edge



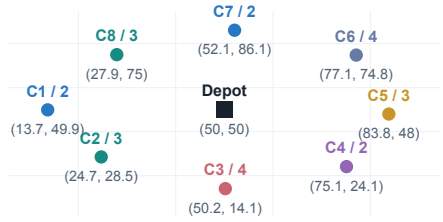
Task: Assign colors to vertices so adjacent vertices differ. Minimize the number of colors used.

mmobgen_gc_easy_000001

(f) Vehicle routing

2 vehicles | capacity 14 each

Labels: customer / demand / (x, y)



Task: Start and end at the depot, visit every customer once, and respect vehicle capacity. Minimize total travel distance.

mmobgen_vrp_easy_000001

Figure 9: Six generated examples pairing visual data with abbreviated modeling tasks. Input data are redrawn as vector graphics for legibility; the dataset images are unchanged. Examples are selected for readability and family coverage, not model performance. No solutions or optimal values are shown.

Table 6: The 7,648 generated training instances by problem family.

Family	Subfamilies	Instances
Location, covering, and assignment	Bipartite assignment; facility location; generalized assignment; p-median/center; set covering/partitioning	1,490
Network optimization	Maximum flow; minimum-cost flow; capacitated network design; resource-constrained shortest path; time-expanded multi-period network flow	1,490
Multi-period and system planning	Lot sizing/production planning; workforce/shift scheduling; energy dispatch/unit commitment; inventory routing	1,193
Scheduling and sequencing	Job-shop scheduling; parallel-machine scheduling; flexible hybrid flow-shop scheduling; resource-constrained project scheduling	1,192
Combinatorial and logical	Multidimensional knapsack; graph coloring; set packing; logical-constraint satisfaction	1,192
Routing and tours	Traveling salesperson; vehicle routing; heterogeneous-fleet vehicle routing; traveling salesperson with time windows	1,091
Total	26 subfamilies across three difficulty levels	7,648

for solution-revealing values or highlights that these scans may miss. §C.6 describes duplicate and overlap checks.

Gurobi training targets. For SFT, we prepare self-contained Gurobi code and check its Python syntax, permitted operations, execution status, and agreement with the reviewed reference result. Instances with unresolved code errors, inconsistent results, or missing required execution evidence are excluded from MLLM4OR-Train.

Interpreting execution results. A successful run with the expected objective value does not prove that the program models the intended problem: a missing constraint may leave that value unchanged. Likewise, text, images, and code generated from one configuration can share the same error. Experts therefore check the problem specification against the formulation and code, as well as reviewing execution results. Corrections follow the intended problem rather than changing it to fit a solver output.

C.6 Deduplication and Dataset Splits

Removing duplicates and conflicting inputs. We compare complete reconstructed English problem statements after normalizing case and whitespace, and compare model inputs using both problem text and image content. Matching problems with consistent reference answers retain one representative; identical inputs with unresolved conflicting answers are excluded. When consistent duplicates span splits, we retain a test or development representative rather than a training copy.

Splitting the reconstructed problems. The OptMATH and OR-Instruct branches each target an 80/10/10 train/dev/test split. OptMATH keeps the coarse screening groups described in §C.2 together and balances problem families and difficulty levels. OR-Instruct groups questions with identical wording after number normalization, together with retrieved near-template matches whose character n-gram TF-IDF cosine similarity is at least 0.88. Each connected group stays in one split, with assignment balancing problem family, visual layout, and the extent of reconstruction repairs. No group defined by these procedures crosses splits. These checks do not rule out all shared templates, and the OptMATH screening groups are not semantic-equivalence classes.

Table 7 reports the final split sizes.

Placement of generated problems. All retained generated problems belong to MLLM4OR-Train, so MLLM4OR-Test contains only reconstructed source problems. Auxiliary development probes are excluded from the corpus counts. Test instances are not used for SFT training or DPO preference construction.

C.7 Final Corpus Composition

The final MLLM4OR-Corpus contains 22,107 problems: 19,189 for training, 1,453 for development, and 1,465 for testing. Table 7 breaks these counts down by source. All four SFT variants use the same 19,189 training problems.

Table 7: Final MLLM4OR-Corpus counts by source and split. Auxiliary development probes are excluded.

Source	Total	MLLM4OR-Train	MLLM4OR-Dev	MLLM4OR-Test
OptMATH-Train	12,374	9,877	1,243	1,254
OR-Instruct-3K	2,085	1,664	210	211
Parametric generators	7,648	7,648	0	0
Total	22,107	19,189	1,453	1,465

Problem and visual coverage. Table 8 summarizes the problem categories and visual layouts represented in the final corpus. These categories are defined separately for each source, so their counts are not a common measure of diversity or difficulty. §C.2–§C.4 explain the source-specific labels and distributions.

Table 8: Problem and visual coverage of the final corpus, using the same retained instances as Table 7. Layout categories are source-specific.

Source	Problem categories	Visual layouts
OptMATH-Train	14 problem families	98 rendering layouts, grouped into 8 display categories
OR-Instruct-3K	14 problem families	17 layout categories
Parametric generators	6 families and 26 subfamilies	41 layout categories

C.8 Corpus Statistics and Reference-Model Sizes

This subsection explains the statistics in Figure 2 and the source-specific distributions in Figures 6–8. All use the final 22,107-instance corpus in Table 7, pooling its training, development, and test splits. Each problem is counted once per distribution; excluded records, auxiliary development probes, and alternative SFT answers are not additional instances.

Source-specific distributions. Each source-level panel uses that source's retained instances as its denominator. OptMATH's 98 rendering layouts are grouped into eight visual formats, whereas OR-Instruct retains its 17 layout categories. These source-specific categories describe visual composition, not a shared measure of diversity. The generator's subfamily panel shows counts for each generation difficulty level.

Application contexts: panel (a). We assign application contexts using a fixed keyword and phrase inventory applied to the reconstructed English problem statement. Matching ignores case and counts distinct complete words or phrases. The category with the most matching terms is selected; ties and statements with no matches are labeled *Unspecified or mixed*. These are keyword-based descriptions, not manually verified industry labels. Table 10 reports the counts.

Problem families: panel (b). We map the reviewed source labels to eight corpus-wide groups using Table 9. OptMATH and OR-Instruct use their final problem-family labels; generated instances use their recorded subfamilies. Each instance enters exactly one group. Unlike the application-context labels, these groups are not inferred from keywords. They combine related optimization structures across sources and differ from the six-family classification used for the generator branch.

Reference models: panels (c)–(e). Formulation classes and model sizes describe the reference optimization programs, not the evaluated models' generated answers. For training instances, we use

Table 9: Mapping source-specific problem labels to the eight corpus-wide groups in Figure 2(b). Each instance belongs to one group.

Group	Included source families or generator subfamilies
Network and routing	Network flow/transport; routing/TSP/VRP; capacitated network design; maximum flow; minimum-cost flow; resource-constrained shortest path; time-expanded multi-period network flow; traveling salesman; TSP with time windows; vehicle routing; heterogeneous-fleet VRP; inventory routing
Production and resources	Production planning; diet/blending; energy/power; energy dispatch/unit commitment; lot-sizing production planning
Scheduling and staffing	Scheduling/sequencing; staffing/shift; job-shop, parallel-machine, and flexible hybrid flow-shop scheduling; resource-constrained project scheduling; workforce planning/shift scheduling
Location and coverage	Facility location; set covering; multi-covering; p-median/center; set covering/partitioning
Selection and investment	Knapsack/selection; portfolio/investment; multidimensional knapsack
Assignment and matching	Assignment/matching; bipartite assignment; generalized assignment
Packing and logic	Cutting/packing; set packing; graph coloring; logical-constraint satisfaction
Other structures	The recorded other/general optimization family

Table 10: Application-context and problem-family counts for Figure 2(a,b). Each count column sums to 22,107.

Application context	Count	Optimization family group	Count
Unspecified or mixed	8,652	Network and routing	6,699
Mobility and logistics	6,864	Production and resources	4,048
Manufacturing and operations	4,138	Scheduling and staffing	3,572
Energy and utilities	828	Location and coverage	2,876
Finance and commerce	686	Selection and investment	1,764
Health and nutrition	669	Assignment and matching	1,553
Public services and environment	216	Packing and logic	1,431
Computing and communications	54	Other structures	164
Total	22,107	Total	22,107

the solver code in the 19,189 Direct-SFT reference answers. For the 2,918 development and test instances, we use their recorded reference programs.

We record the Gurobi model at the first optimization call, before presolve or optimization. Later additions or refinements are not included. This measures the reference formulation's structure, separately from the execution checks used to verify its answer.

Formulation classes: panel (c). We classify linear models by their variable domains: continuous LP if all variables are continuous; binary linear if all are binary; integer linear if all are discrete but not all binary; and mixed-integer linear if both continuous and discrete variables are present. Integer variables bounded within $[0, 1]$ count as binary.

A quadratic objective gives QP or MIQP, depending on whether discrete variables are present. Any quadratic constraint instead gives QCP or MIQCP, taking precedence over the objective type. The classification describes the constructed model, even when a different formulation could be linear. No reference model contains general or SOS constraints. Figure 2(c) combines the 163 quadratic models into one slice; Table 11 lists their separate classes.

Model sizes: panels (d,e). We count all variables explicitly created by the reference program, including auxiliary variables. Constraint counts include linear, quadratic, general, and SOS constraints, with each submitted constraint counted once. Variable bounds and integrality declarations are not additional constraint rows. Solver-generated cuts and presolve transformations are excluded.

The reference models contain 1–8,701 variables (median 35) and 0–5,204 constraints (median 24). The 28 models with no explicit constraint rows may still impose bounds or restrictions on variable

Table 11: Formulation classes of the 22,107 reference optimization models. Figure 2(c) combines the three quadratic classes into one slice.

Reference formulation class	Count
Binary linear	7,720
Mixed-integer linear	7,642
Continuous linear (LP)	4,201
Integer linear, not all binary	2,381
Continuous quadratic objective (QP)	158
Mixed-integer quadratic objective (MIQP)	4
Mixed-integer quadratically constrained (MIQCP)	1
Total	22,107

domains. These counts describe the chosen reference formulations, not an intrinsic size or difficulty of the underlying problems.

Both histograms show instance counts on a logarithmic horizontal axis. The 28 zero-constraint models are annotated separately because zero cannot appear on that axis. Percentages use the final corpus size of 22,107 and are rounded to one decimal place.

C.9 Corpus Scope and Limitations

Coverage. The corpus contains problems that can be reconstructed or generated with clear text–image inputs. Training instances must also have verified reference code. This selection does not yield a random sample of real-world optimization problems. Its programmatically rendered visuals do not cover the full variability of workplace documents, such as noisy scans or changing dashboard states.

Interpreting the evidence. The corpus statistics describe the retained problems and their reference formulations. They do not measure how much models rely on images or how much each source contributes to training performance. Human review and solver checks reduce errors but do not guarantee an error-free corpus.

D Evaluation Datasets

This appendix describes the evaluation datasets. We summarize the test suites and development set (§D.1), then describe MLLM4OR-Test and MM-OptBench (§D.2) and the selection, reconstruction, and verification of LLM4OR-VL (§D.3–§D.5). We conclude with its input formats and statistics (§D.6–§D.7).

D.1 Test Suites and Development Set

All three test suites require a mathematical formulation and executable solver code, but differ in their problem sources and input construction. MLLM4OR-Test contains held-out OptMATH and OR-Instruct problems prepared through the same reconstruction pipeline as their training counterparts. MM-OptBench provides an existing multimodal benchmark for the same task; we use its inputs without reconstruction. LLM4OR-VL contains problems reconstructed from eight text-only sources, extending the range of problem sources and visual formats.

Table 12 summarizes the three test suites and the development set used for model selection. Development scores are excluded from test aggregates. We report per-suite results and source-level breakdowns for LLM4OR-VL because an aggregate can hide differences across datasets. Their relative sizes reflect dataset construction, not the prevalence of optimization tasks in practice.

D.2 MLLM4OR-Test and MM-OptBench

MLLM4OR-Test. The 1,465-instance corpus test split contains 1,254 problems reconstructed from OptMATH and 211 from OR-Instruct, with no generated instances. Each problem has a reference

Table 12: Test suites and development set. Counts refer to problem instances.

Dataset	Instances	Purpose	Scope
MLLM4OR-Test	1,465	Evaluation on held-out corpus problems	OptMATH and OR-Instruct reconstructions; no generated instances
MM-OptBench	780	Evaluation on an existing multimodal benchmark	Six optimization families, 26 subcategories, and three difficulty levels
LLM4OR-VL	878	Evaluation across eight reconstructed text-only sources	Selected for coverage; not restricted to previously unseen problem families
MLLM4OR-Dev	1,453	Model selection (§B.3)	Complete development split; excluded from test aggregates

formulation, solver program, and verified result. Groups formed by the source-specific splitting rules stay within one split (§C.6), although similar problem templates may appear in different splits. This suite evaluates held-out problems from the same reconstructed sources used for training.

MM-OptBench. This benchmark contains 780 text–image problems spanning six optimization families, 26 subcategories, and three difficulty levels (Li et al., 2026b). We use its original inputs and reference answers without reconstruction and evaluate generated code using the procedure in §B.2.

Examples across the test suites. Figure 10 shows three examples from each of the three test suites. Panels (a)–(c) show MM-OptBench routing, production-planning, and energy tasks; panels (d)–(f) show MLLM4OR-Test aircraft-assignment, portfolio-selection, and health-care-staffing tasks; panels (g)–(i) show LLM4OR-VL diet, telecommunications, and crop-planning tasks. The examples illustrate different applications and visual formats and are not selected by model performance.

D.3 LLM4OR-VL Sources and Selection

LLM4OR-VL contains 878 text–image optimization problems used only for evaluation. They are reconstructed from eight source collections: ComplexOR (Xiao et al., 2024), IndustryOR (Huang et al., 2025), MAMO Easy LP and Complex LP (Huang et al., 2024), NLP4LP (AhmadiTeshnizi et al., 2024), NL4Opt (Ramamonjison et al., 2023), ReSocratic (Yang et al., 2025), and OptMATH-Bench (Lu et al., 2025). These collections contain 1,677 source problems in total. We leave 436 MAMO Easy LP problems outside the selected sample and remove 363 duplicates, retaining 878 problems for reconstruction. Table 13 gives the counts for each source.

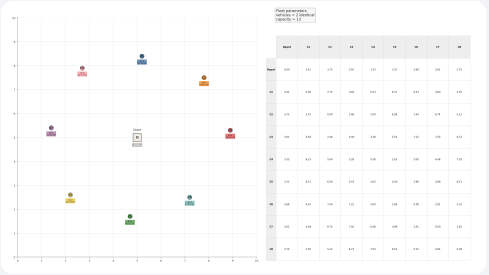
Table 13: Source selection for LLM4OR-VL. The 436 unsampled problems and 363 removed duplicates reflect sampling and deduplication, not rejection for poor quality.

Source	Source pool	Not sampled	Duplicates	Retained
ComplexOR	18	0	0	18
IndustryOR	42	0	0	42
MAMO Easy LP	545	436	0	109
MAMO Complex LP	111	0	5	106
NLP4LP	178	0	0	178
NL4Opt	214	0	147	67
ReSocratic	403	0	211	192
OptMATH-Bench	166	0	0	166
Total	1,677	436	363	878

Sampling MAMO Easy LP. We select 109 of its 545 problems (20%) using a deterministic procedure that prioritizes underrepresented problem structures, application contexts, and objective directions. The 20% fraction applies to the whole source pool, not to each problem family. This is

(a) Routing and logistics

MM-OptBench

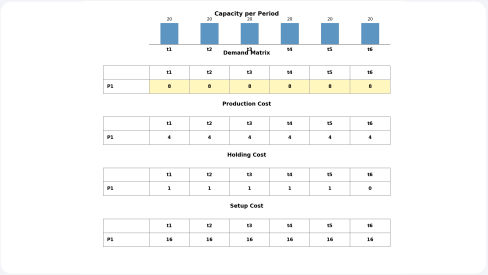


Task: Choose depot-to-depot routes visiting every customer once, within vehicle capacity.

vrp_e_001

(b) Production planning

MM-OptBench

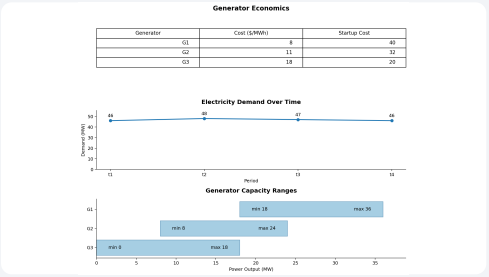


Task: Meet demand using capacity-limited production and inventory, with no backlog.

lspp_e_001

(c) Energy scheduling

MM-OptBench

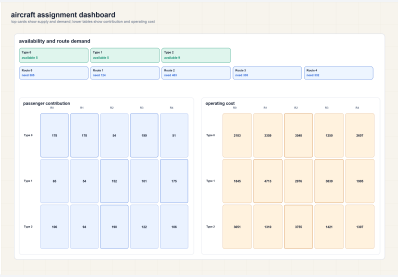


Task: Choose generator states and output to meet demand exactly, including startup costs.

educ_e_001

(d) Aircraft assignment

MLLM4OR-Test / OptMATH

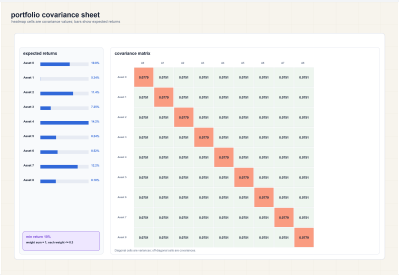


Task: Assign whole aircraft to routes to meet demand at minimum operating cost.

optmath_train_038910

(e) Portfolio selection

MLLM4OR-Test / OptMATH

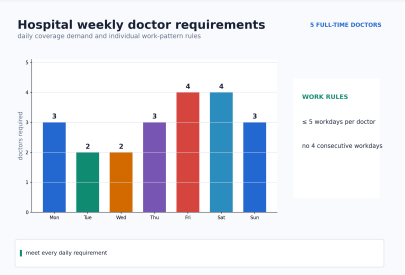


Task: Minimize variance subject to the return target, full investment and asset limits.

optmath_train_113299

(f) Health-care staffing

MLLM4OR-Test / OR-Instruct



Task: Cover daily doctor requirements under weekly workload and consecutive-day limits.

or_instruct_000501

Figure 10: Original test inputs and abbreviated modeling tasks: (a)–(c) MM-OptBench (Li et al., 2026b) and (d)–(f) MLLM4OR-Test. Images are embedded at full source resolution, without cropping or redrawing. No reference solutions are displayed. Panels (a)–(c) illustrate the settings discussed in the Introduction; panels (g)–(i) continue on the next page.

(g) Diet planning

LLM4OR-VL / ComplexOR



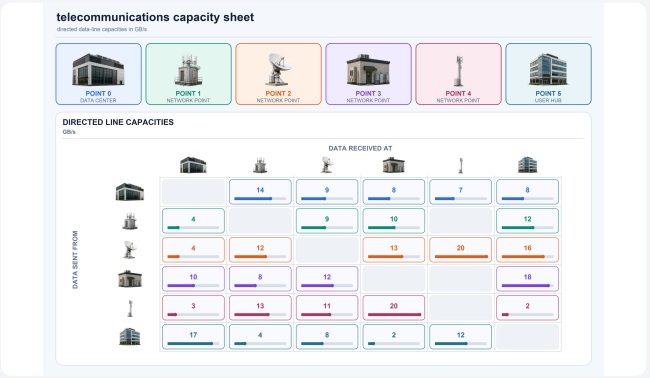
Task
Choose fractional food quantities within purchase bounds to meet nutrient ranges at minimum cost.

Visual input
Food prices, nutrient contributions, purchase bounds and nutrient requirements.

complexor_000006

(h) Telecommunications

LLM4OR-VL / MAMO



Task
Maximize transfer from data center 0 to user hub 5, respecting directed-link capacities and flow conservation.

Visual input
Node identities and every directed data-line capacity.

mamo_complex_lp_000051

(i) Crop planning

LLM4OR-VL / ReSocratic



Task
Choose whole-ton planting quantities to maximize net profit per labor hour under seasonal limits and minimum planting requirements.

Visual input
Crop prices, seed costs, labor requirements and shared seasonal limits.

resocratic_000153

Figure 10 (continued): Original LLM4OR-VL inputs for (g) diet planning, (h) telecommunications, and (i) crop planning. Wider panels retain the complete images alongside task summaries and the information conveyed visually. Source suites and instance IDs identify each example.

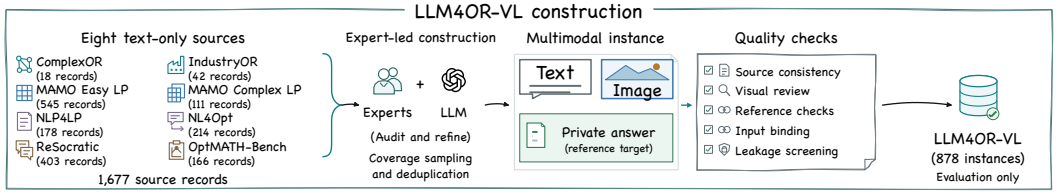


Figure 11: Construction of LLM4OR-VL. Sampling and deduplication retain 878 of 1,677 source problems. Experts, assisted by LLMs, reconstruct the text–image inputs and check them together with the private reference answers.

a coverage-oriented selection rather than a random sample, so results describe the selected subset rather than the original source distribution.

Removing duplicates. We remove repeated problems within and across source collections using exact matches and expert review of requirements and numerical data. These checks include NL4Opt prompts already represented in NLP4LP. Table 13 reports the removed duplicates by source. Distinct retained problems may still share a mathematical template.

D.4 Problem Reconstruction and Text–Image Design

Preparing the problem statement. We start from the original question, its answer, and any available structured data or code. We identify the problem data, decisions, and operational rules, remove formulation or implementation hints, and clarify missing or inconsistent information. Original statements and answers are retained alongside the revisions.

Handling missing or rounded data. Where a source is incomplete, we record any added assumptions rather than treating them as recovered source data. Values rounded for readability must agree across the text and image, and reference answers are recomputed when those changes affect the solution. Each reconstructed problem is evaluated against its own requirements and reference answer, not automatically against the original source answer.

Dividing information between text and image. The image presents numerical data or relationships through networks, matrices, timelines, plots, or other labeled displays. The text states the scenario, decision objective, and rules not shown clearly in the image. Together, they must fully specify the reconstructed problem without revealing its formulation or solution. For example, Figure 10(h) shows directed link capacities in a matrix, while the text asks for the maximum flow. The image thus conveys problem data rather than simply reproducing the original paragraph.

Model inputs and reference answers. For each problem, the model receives the complementary text and rendered image. Reference answers, revision notes, verification calculations, and any available reference code are excluded from its input. The evaluation package contains private reference answers for all 878 instances, but does not include a reference formulation or solver program for every instance. Evaluation executes the model-generated program and checks its result against the reference answer, as described in §B.2.

D.5 Quality Control and Dataset Overlap

Review team. The same six experts described in §C.1 review LLM4OR-VL. Two reviewers assess each of the 878 retained problems, and a third resolves ambiguous cases or substantive disagreements. They examine the source question and answer, the reconstructed requirements, the text–image input, and the evidence supporting the reference answer. Figure 11 summarizes the checks.

Input checks. Automated checks verify that each problem has readable input files and the intended text–image pair. Experts check values, labels, units, arrow directions, and index ranges against the reconstructed requirements. They also check readability, completeness across the two modalities, and

the absence of exposed answers or construction notes. Duplicate review compares problem data and requirements, not visual similarity alone (§D.3).

Reference-answer checks. Reviewers examine analytical derivations, enumeration, optimization runs, or source answers with supporting review notes, as available. They check that each answer applies to the reconstructed problem, including added assumptions and rounded data (§D.4). This construction-stage review is separate from the execution-based scoring of model-generated code (§B.2).

Corrections and rechecking. Unclear or incorrect inputs are revised and reviewed again for readability, text–image consistency, and answer leakage. Changes to problem data or requirements also require the reference answer to be checked again. Review reduces errors but does not establish that the suite is error-free.

Estimated review workload. Two reviews per problem give 1,756 assessments. At an estimated two minutes each, routine review takes about 59 person-hours ($878 \times 2 \times 2/60 \approx 58.5$). This is an estimate, not recorded working time; it excludes construction, rendering, prior verification, rework, adjudication, and automated runtime.

Overlap with training data. Evaluation and training instance IDs are disjoint, but this does not establish that their content is disjoint. The datasets may share source material or optimization structures, and exposure during base-model pretraining has not been audited. We therefore do not claim that LLM4OR-VL is free of training-data overlap or consists of wholly unseen problem families.

D.6 Multimodal and Text-Only Inputs

Each of the 878 problems in LLM4OR-VL has two input formats. The multimodal format pairs an image with complementary text and is used for model evaluation. The text-only format, called *oracle text*, provides the complete English problem statement without an image. It uses the revised statement when available and the original statement otherwise (§D.4). Both formats share the same instance ID and private reference answer; the answer is not given to the model. The complete text also supports the application-context statistics below (§D.7).

D.7 LLM4OR-VL Input Statistics

Figure 12 summarizes the 878 retained LLM4OR-VL problems. Each has one English complementary prompt and one image. All five panels describe this same set, rather than the original source pools. Percentages use 878 as the denominator and are rounded to one decimal place.

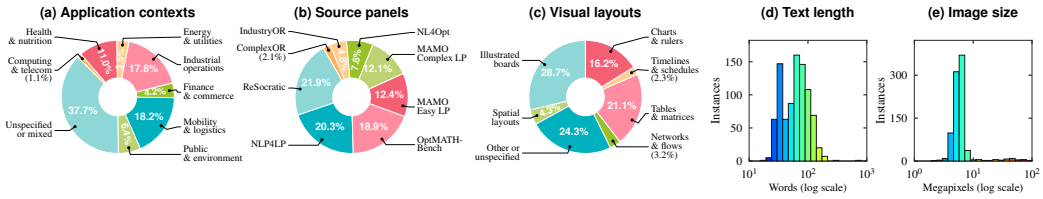


Figure 12: Input statistics for all 878 LLM4OR-VL problems: (a) application contexts, (b) source datasets, (c) visual-layout categories, (d) complementary-prompt length, and (e) original image size. Contexts and layouts use rule-based categories. Both histograms show instance counts in logarithmically spaced bins on logarithmic horizontal axes.

Application contexts: panel (a). We apply the keyword-based classification from §C.8 to the complete English problem statement in the oracle-text view. That statement is used only to assign descriptive labels; it is not added to the multimodal evaluation input (§D.6). The category with the most matching terms is selected, while ties and statements with no matches are labeled *Unspecified or*

mixed. Mobility and logistics (18.2%), industrial operations (17.8%), and health and nutrition (11.0%) are the largest named categories. The 37.7% labeled *Unspecified or mixed* reflects the classification rule, not missing problem statements. These labels are descriptive summaries, not manually verified industry classifications.

Source datasets: panel (b). The source counts match Table 13. ReSocratic (21.9%), NLP4LP (20.3%), and OptMATH-Bench (18.9%) contribute the largest shares. These proportions describe the suite after selection and deduplication, not equal sampling from the eight sources or the prevalence of these problems in practice.

Visual layouts: panel (c). We group each image using keywords in its recorded layout description, with a fixed priority order when multiple categories match. These are metadata-based categories, not manual judgments of the dominant visual layout. Illustrated boards (28.7%), tables and matrices (21.1%), and charts and rulers (16.2%) are the most common named categories. Unmatched or missing descriptions fall under *Other or unspecified* (24.3%). Each image belongs to one group for counting, although it may combine several visual elements. Table 14 lists the context and layout counts.

Table 14: Application-context and visual-layout counts for Figure 12(a,c). The two column pairs are separate distributions, not context–layout combinations; each count column sums to 878.

Application context	Count	Visual layout	Count
Unspecified or mixed	331	Illustrated boards	252
Mobility and logistics	160	Other or unspecified	213
Industrial operations	156	Tables and matrices	185
Health and nutrition	97	Charts and rulers	142
Public services and environment	56	Spatial layouts	38
Finance and commerce	37	Networks and flows	28
Energy and utilities	31	Timelines and schedules	20
Computing and communications	10		
Total	878	Total	878

Text length and image size: panels (d,e). Complementary prompts contain 17–799 words, with a median of 64.5. We count words separated by whitespace, excluding system instructions, image placeholders, and output-format instructions. These are word counts, not model-token counts.

Image area is width times height divided by 10^6, in megapixels. Original images range from 2.32 to 81.65 megapixels (median 6.08), before model-specific processing. These file measurements do not describe inference-time resolution or visual-token usage. Neither image area nor prompt length directly measures mathematical problem complexity.

E SFT Data and Supervision

This appendix describes how we prepare SFT reference answers (§E.1) and assign answer formats in Mixture-SFT (§E.2). Training and evaluation settings are given in §B.3.

E.1 SFT Reference Answers

All four SFT settings use the same 19,189 MLLM4OR-Train problems. Direct-SFT answers contain only a mathematical formulation and solver code. Compact-SFT and Detailed-SFT add concise and detailed explanations, respectively, before these outputs. Mixture-SFT assigns each problem either a Direct-SFT or a Compact-SFT answer, with one answer per problem. Table 15 summarizes these formats.

Preparing explanations. To prepare the explanations, we provide frontier LLMs with the text–image input, the complete problem statement, and a verified mathematical formulation. We ask them to explain how specific information in the text and image supports the choice of variables, objective,

Table 15: SFT reference-answer formats. Every answer contains a formulation and solver code, preceded by an explanation when specified.

Setting	Explanation in the answer	Comparison of interest
Direct-SFT	None	Learning from formulations and code alone
Compact-SFT	Concise structured explanation	Adding links from input evidence to modeling decisions
Detailed-SFT	Detailed structured explanation	Increasing explanatory detail
Mixture-SFT	Concise explanation for a fixed subset; none otherwise	Varying the fraction of problems with explanations

and constraints. These explanations are written for training; they are not records of a model's internal reasoning. Qwen3.5-9B is trained to generate the reference answer from the text–image input and a format instruction. The formulation and code belong to the training target, not the problem prompt.

The annotation instructions prohibit optimal objective values, solved decision-variable assignments, source answers, and correction notes. Automated validation checks required fields, references to supporting text or image regions, and potential answer leakage. Reference programs are checked separately by execution.

Detailed and concise formats. Detailed-SFT explanations have six components: visual facts, sets and parameters, decision variables and domains, the objective, constraints, and a solver-implementation plan. Evidence references identify the relevant text passage or image location and the modeling decision it supports.

To obtain Compact-SFT explanations, we shorten the detailed annotations using fixed rules. We keep the modeling components and the first cited text or image location for each item, but remove the separate list of visual facts and generic implementation steps. The resulting fields describe the problem, input data, variables, objective, constraints, and implementation plan. Both formats explain how modeling choices follow from the input, but Compact-SFT includes less supporting detail.

An example across formats. Consider a cutting-pattern problem in which the text requires meeting each width's demand and a matrix gives the output of each cutting pattern. Direct-SFT provides the formulation and code. Compact-SFT first explains how the demand requirement and matrix entries determine the constraints. Detailed-SFT adds more specific references to the text and image and a fuller implementation plan. The formulation and code are identical across these three answers. The explanations describe how to build the model, without giving the optimal pattern counts or objective value.

Training instructions and loss. Each training prompt requests the assigned answer format. Direct-SFT requests only a formulation and code; Compact-SFT and Detailed-SFT also request an explanation before them. Mixture-SFT uses the instruction corresponding to each problem's assigned format (§E.2). The problem text and image are shared across settings, but the format instructions differ.

The next-token loss covers the complete reference answer, including any explanation. The problem text, image, and instructions provide context and are excluded from the loss. At evaluation, all variants receive the same instruction, requesting only the formulation and code in one call (§3.4.1); there is no separate explanation-generation stage. Using the same training problems does not equalize answer lengths, supervised-token counts, or computational cost.

E.2 Mixture-SFT Construction

Mixture-SFT combines two answer formats: Direct answers contain a formulation and code, while Compact answers add a concise explanation before them. We express each mixture as a Direct:Compact ratio by problem count and compare 30:70, 40:60, 50:50, 60:40, and 70:30. These are problem ratios, not token ratios. Unless a ratio is specified, Mixture-SFT refers to the 70:30 variant.

One answer per problem. Every variant uses the same 19,189 training problems. The two candidate answers for each problem share the same formulation and code. Each problem is assigned

one answer format and the instruction requesting that format. We include only the assigned answer, not both available answers, and keep the assignment fixed throughout training. Changing the ratio therefore changes how many problems include explanations, not the number of training problems.

Assigning answer formats. We group problems by source, optimization family, difficulty, and visual layout. Within each group, we alternate between clusters of similar reference programs, then combine the group-specific orders in approximate proportion to their sizes. This produces one fixed ordering for all five mixtures and limits the concentration of similar code structures.

For each ratio, we assign Direct answers to the required number of problems at the start of this ordering and Compact answers to the rest. The Direct subsets are therefore nested: every problem assigned a Direct answer at 30:70 keeps that format at 40:60, and so on. Table 16 lists the counts after rounding to whole problems.

Table 16: Mixture-SFT composition by problem count. Every ratio uses the same 19,189 problems, with one assigned answer per problem.

Direct:Compact	Direct answers	Compact answers
30:70	5,757	13,432
40:60	7,676	11,513
50:50	9,595	9,594
60:40	11,513	7,676
70:30	13,432	5,757

F Solver-Grounded Direct Preference Optimization

This appendix describes how we construct preference pairs and train SG-DPO. We explain the method and verification rules (§F.1–§F.2), then describe response generation, pair construction, and training (§F.3–§F.5). §F.6 states the evaluation protocol and its interpretation limits. Results are reported in §4 and §G.

F.1 Rationale and Training Procedure

SG-DPO trains an SFT model to prefer a verified reference answer over its own incorrect answer to the same text–image problem. A program can run successfully and still solve the wrong problem, so execution alone is not enough to judge the answer. We also check the solver status and compare the returned objective with the reference value. Figure 4 summarizes the three steps: checking generated answers, constructing preference pairs, and updating the model.

Distinguishing failure types. The checks assign labels F0–F4 according to where verification stops. F0 and F1 cover format, safety, and execution failures; F2 denotes a program without an accepted optimal finite result; F3 denotes an accepted optimal result with the wrong objective value. F4 means that the result agrees with the reference under these checks, not that the formulation has been proved equivalent. Table 17 gives the full definitions and eligibility rules.

These labels separate basic output and execution problems from errors that remain after the solver reports an optimum. They do not rank every error by difficulty or determine its cause. We retain distinct eligible failures rather than selecting only the highest-ranked failure for each problem.

An example. Suppose the text asks to maximize throughput q and the image gives a capacity of 5. The reference problem is $\max q$ subject to $0 \le q \le 5$. A generated program that uses $q \le 8$ can solve successfully and return 8, but receives F3 because it disagrees with the reference. A program that stops with a Python name error receives F1. This example illustrates the checks; training uses the model's actual responses, not errors inserted into correct programs.

Constructing same-problem pairs. For each eligible failed response y^-, the positive response y^+ is the verified reference formulation and code for that same input. The failed response is kept

unchanged. This provides a training pair even when both sampled answers are wrong. An F4 response is recorded as a success but does not replace the verified reference answer. Private objective values and verification reports are used to assess responses, not appended to the model-facing input.

Failure labels alone do not make a response eligible for training. F1 negatives must have valid output sections, valid syntax, and statically safe code; F0 negatives require a separate static-safety check. Unsafe programs are never executed, and infrastructure or verification exceptions are excluded from the failure pairs. Duplicate pairs are removed, and both complete input–response sequences must fit the training context without truncation (§F.4).

Giving each problem equal weight. A problem can contribute two distinct failures and therefore two pairs. Without weighting, it would count twice as much as a problem contributing one pair. Let N be the number of retained pairs, M the number of represented problems, and m_x the number of pairs for problem x. Each pair receives multiplier $N/(Mm_x)$ when averaging the standard DPO loss. Thus each problem has the same total weight in the dataset objective, regardless of how many pairs it contributes. This equalizes problem weights, not source weights.

The policy and frozen reference both start from the same Mixture-SFT checkpoint. We update only the policy, using the fixed preference pairs and the loss in §3.4.2. No new model responses or solver calls are needed during these updates. Algorithm 1 summarizes the procedure.

Algorithm 1 Solver-grounded DPO with problem-normalized pair weights

Inputs: SFT policy π_0; training inputs with verified reference answers and private reference outcomes; context limit L.
Output: preference-trained policy π_θ.

1. **Generate responses.** Sample two answers from π_0 for each training input.
2. **Check responses.** Check format, syntax, and static safety. Execute admissible programs, check solver status, and compare results with the same-input reference. Assign F0–F4; keep infrastructure or verification exceptions separate.
3. **Form pairs.** Pair each eligible F0–F3 response y^- with the same-input verified answer y^+. Record F4 responses separately, without using them as positive targets.
4. **Filter pairs.** Remove identical chosen/rejected answers and duplicate or redundant candidates. Keep distinct eligible failures, including both failures from one input. Check both complete sequences with the training tokenizer and image processor; exclude overlength pairs without truncation.
5. **Set weights.** Freeze the retained dataset $\mathcal{D}_{\text{pref}}$. Count its N pairs, M distinct inputs, and m_x pairs per input. Assign multiplier $N/(Mm_x)$ to each pair.
6. **Train.** Initialize π_θ and π_{ref} from π_0. Keep π_{ref} frozen and minimize the weighted mean of the per-pair DPO loss (§3.4.2).

F.2 Response Verification and Pair Eligibility

We sample the 2,000 problems used to generate preference candidates from the 19,189-instance MLLM4OR-Train (§F.3). Development data, auxiliary development probes, and all three test suites are excluded from this sampling and from preference training. Each generated answer is checked against the private reference for the same text–image input. Reference outcomes are used only for verification; they are not added to the prompt or the rejected answer.

Table 17 defines the response labels and the conditions for using an answer as the negative response y^-. A failure label alone is not enough: F1 includes syntax errors, but only F1 answers with valid output sections, valid Python syntax, and code that passes the static-safety checks are eligible. Unsafe code is neither executed nor used as a negative. Failures of the execution infrastructure or verification system remain unresolved and are excluded from preference pairs.

For scalar objectives, we use absolute tolerance 10^{-6} and relative tolerance 10^{-5}, following the numerical comparison defined in §B.2. These labels describe the verification outcome, not the underlying modeling error. F3 may result from an incorrect coefficient, constraint, variable domain, or implementation. F4 confirms agreement under the checks but does not prove that the generated formulation is equivalent to the reference.

Table 17: Response verification and eligibility as a negative answer. Eligible candidates must also pass the pair-level checks in §F.4.

Label	Verification outcome	Eligible as y^-?
F0	Missing or invalid output sections, or failed static-safety checks	Format failures only, after a separate static-safety check
F1	Syntax error, execution failure (including timeout), or invalid result format	Only with valid sections, valid syntax, and passed static-safety checks
F2	Execution completes, but an accepted optimal status or a finite objective is missing	Yes
F3	An accepted optimal, finite objective differs from the reference beyond tolerance	Yes
F4	All checks pass, including objective agreement within tolerance	No; record as a success only

F.3 Generating Candidate Responses

We generate candidate responses with the Mixture-SFT checkpoint that also initializes the DPO policy and frozen reference model. We select 2,000 distinct training problems: 667 from the generator branch, 667 from OptMATH, and 666 from OR-Instruct.

For each text–image problem, we sample two answers at temperature 0.7 and top-$p = 0.95$, allowing up to 16,384 new tokens per answer. The image-token limit is also 16,384. We retain all 4,000 generated answers and apply the checks in §F.2. Only programs that pass the format, syntax, and static-safety checks are executed.

F.4 Constructing Preference Pairs

Each preference pair compares two answers to the same text–image problem: the verified reference answer is the positive response y^+, and an eligible failed answer generated by Mixture-SFT is the negative response y^-. We keep the generated answer unchanged. If both generated answers pass verification (F4), the problem contributes no pair. If only one fails and is eligible, we pair it with the reference answer, not with the successful generated answer.

We remove pairs with identical positive and negative answers, duplicate pairs, and redundant responses to the same problem. Each pair is encoded as two separate sequences, combining the input with either the positive or the negative answer. Both sequences must fit the 16,384-token training context, including the image tokens, input text, and answer. We exclude overlength pairs rather than truncate either sequence.

We retain both distinct eligible failures from a problem and do not impose quotas by source or program-template group. The per-problem weights in §F.1 prevent a problem with two pairs from receiving twice the total weight of one with a single pair.

F.5 DPO Training Configuration

We minimize the DPO loss in §3.4.2, using the per-problem weights described in §F.1. For both answers in a pair, we sum log-probabilities over the supervised response tokens. The problem text, image, and output instructions provide context but are excluded from these sums.

We update only the language model's LoRA parameters. The base model weights, vision encoder, multimodal aligner, and reference model remain fixed. Dropout is disabled during DPO. Table 18 summarizes the training settings and distinguishes DPO training hardware from rollout and evaluation hardware.

F.6 Evaluation Protocol and Scope

Checkpoint selection. We select the SG-DPO checkpoint on the complete MLLM4OR-Dev (1,453 instances) from the twelve saved checkpoints. The selected model, checkpoint 65, is then evaluated

Table 18: Training configuration and inference hardware for Mixture-SFT-initialized SG-DPO.

Setting	Value
Policy and reference	Both initialized from Mixture-SFT checkpoint 4798
Preference data	498 pairs from 315 training problems
Loss	Sigmoid DPO; per-problem pair weights
LoRA	Rank 32; $\alpha = 64$; dropout disabled
Optimizer	AdamW; learning rate 5×10^{-7}; weight decay 0.1
DPO coefficient	$\beta = 0.2$
Learning-rate schedule	Cosine decay with 5% warm-up
Precision / seed	bfloat16 / 42
DPO training hardware	One NVIDIA H200 GPU
Rollout / inference hardware	Multiple NVIDIA H100 GPUs
Batch size	One pair per microbatch; four-step gradient accumulation
Training budget	156 optimizer updates
Checkpoint interval	Every 13 updates; twelve checkpoints

on all three complete test suites. Development scores are excluded from test aggregates, and test instances are excluded from preference construction and gradient training.

Evaluation settings. Each test instance receives one greedy answer and four sampled answers, using the shared SFT/DPO sampling settings (§B.3). The output limit is 32,768 tokens per answer, and the image-token limit is 16,384. Generated programs are executed without repair under the shared scoring procedure (§B.2). Pass@1 reports the five-run mean and sample SD; pass@4 uses only the four sampled answers (§B.4).

Sampled-run check. As a diagnostic separate from the primary five-run mean, we recompute mean accuracy and sample SD over only the four sampled runs. On LLM4OR-VL, these are $44.1 \pm 3.6\%$ for Mixture-SFT and $48.3 \pm 1.4\%$ for SG-DPO; on MM-OptBench, they are $37.8 \pm 7.0\%$ and $44.4 \pm 0.9\%$, respectively. Higher means and lower observed variation therefore persist after excluding greedy. This variation concerns inference runs, not independently trained model replicas.

Interpretation limits. The comparison evaluates the DPO procedure as a whole. It does not isolate the contributions of negative-response filtering, per-problem weighting, and additional training; no component ablations are reported.

G Supplementary Results and Analysis

This appendix reports four-answer coverage (§G.1), single-format SFT comparisons (§G.2), and results by problem family, difficulty, and source (§G.3–§G.5). §G.6 analyzes failure types and paired changes after DPO. §G.7 explains the evaluation scope for the mathematical baselines.

Reading the tables. For finalized results, pass@1 cells with an SD give the five-run mean and sample SD; cells without an SD give first-run accuracy. SD describes inference-run variation, not variation across independently trained models. Micro and Macro are computed within each run before summarizing across runs (§B.4).

Result coverage. The four main SFT variants, SG-DPO, and all three mathematical baselines have five-run results on all three test suites. Results for Kimi-K2.5, Gemini 3.1 Pro, and Claude Sonnet 4.6 are provisional values awaiting full-run replacements. A dash marks an unavailable result, not zero accuracy.

Comparison scope. SFT and DPO use the same sampling temperature and top-p (§B.3). This does not imply identical settings for every baseline: the mathematical baselines do not have a verified common image-processing protocol (§G.7). Component-level limits of the DPO comparison are discussed in §F.6.

G.1 FOUR-ANSWER COVERAGE (PASS@4)

Table 19 reports the percentage of instances for which at least one of four additional sampled answers passes the scoring procedure. The first-run response is excluded. Pass@4 measures whether these four candidates contain a successful answer, not whether the model can identify that answer. Each finalized score requires four scored answers per instance; provisional results and dashes follow the reporting conventions above.

Micro weights the three suites by instance count; Macro gives each suite equal weight. The eight LLM4OR-VL source columns are breakdowns of one suite, not additional suites. Pass@4 is a single coverage statistic, so we do not report an across-run SD (§B.4).

Table 19: Pass@4 (%) from four additional sampled answers per instance, excluding the first-run response. Evaluation protocol: §B.2. –: pending. Coverage and provenance: §G.

	Model	MLLM4OR-Test (1,465)	LLM4OR-VL									MM-OptBench (780)	Micro Avg	Macro Avg
			C-OR (18)	I-OR (42)	MAMO-C (106)	MAMO-E (109)	NL4Opt (67)	NLP4LP (178)	OptMATH (166)	ReSoc (192)	Overall (878)			
General MLLMs	Claude Sonnet 4.6	76.3	75.0	0.0	76.2	90.9	69.2	83.3	63.6	92.3	76.4	39.7	67.2	64.1
	Gemini 3.1 Pro	97.1	100.0	87.5	100.0	95.5	76.9	97.2	97.0	100.0	95.9	76.9	91.7	90.0
	GPT-5.5	92.9	88.9	83.3	99.1	100.0	76.1	95.5	97.0	97.9	95.1	75.9	89.3	88.0
	Kimi-K2.5	93.4	75.0	75.0	95.2	95.5	76.9	91.7	93.9	100.0	92.6	67.3	86.6	84.4
	Qwen3.6-Plus	41.7	38.9	35.7	64.2	73.4	32.8	49.4	45.2	53.6	52.2	39.4	44.1	44.4
	GLM-4.5V	74.8	77.8	57.1	78.3	99.1	76.1	79.8	46.4	93.2	77.2	28.6	63.9	60.2
Math MLLMs	MathCoder-VL-8B	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	MM-Eureka	18.6	27.8	28.6	29.2	75.2	22.4	39.9	1.8	53.6	36.7	0.5	19.1	18.6
	MM-PRM	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Qwen3.5 Fine-tuned Models	Qwen3.5-9B	62.5	77.8	54.8	64.2	78.9	61.2	65.7	46.4	75.5	65.0	31.4	55.5	53.0
	Direct-SFT	85.7	77.8	47.6	72.6	97.2	55.2	69.1	22.3	69.8	62.4	54.1	71.3	67.4
	Compact-SFT	79.5	72.2	42.9	72.6	88.1	68.7	78.1	27.7	81.8	67.4	45.3	67.6	64.1
	Detailed-SFT	73.4	50.0	40.5	50.9	85.3	50.7	62.4	22.3	68.8	55.5	46.0	61.5	58.3
	Mixture-SFT	92.9	61.1	52.4	73.6	98.2	55.2	66.3	33.7	67.7	63.7	53.6	74.9	70.1
	SG-DPO	93.2	55.6	54.8	73.6	98.2	56.7	68.5	36.1	67.2	64.6	56.7	76.0	71.5

G.2 SINGLE-FORMAT SFT RESULTS

Table 20 compares the three single-format SFT models on the external suites, reporting five-run mean pass@1 with sample SD and pass@4.

Compact-SFT leads on LLM4OR-VL under both metrics, while Direct-SFT leads on MM-OptBench. Detailed-SFT leads on neither suite. The preferred single format therefore differs by benchmark, but is consistent across the two metrics within each benchmark.

Table 20: Single-format SFT results (%). Each cell gives five-run mean pass@1 (with sample SD) / pass@4. Pass@4 excludes the greedy response. Metrics: §B.4; scoring: §B.2.

Model	LLM4OR-VL	MM-OptBench
Direct-SFT	$41.0_{\pm 0.9}$ / 62.4	$40.8_{\pm 1.3}$ / 54.1
Compact-SFT	$44.8_{\pm 0.9}$ / 67.4	$29.7_{\pm 1.2}$ / 45.3
Detailed-SFT	$29.2_{\pm 1.7}$ / 55.5	$29.3_{\pm 1.1}$ / 46.0

G.3 MM-OPTBENCH RESULTS BY FAMILY AND DIFFICULTY

Tables 21 and 22 report pass@1 and pass@4, respectively, grouped by optimization family and by benchmark-defined difficulty. Both groupings cover the same 780 test instances. The family names are shortened in the tables: Location includes covering and assignment, Scheduling includes sequencing, Planning denotes multi-period and system planning, and Logical denotes combinatorial and logical problems. Column headers give group sizes. Overall pools all instances rather than weighting the six families equally; each difficulty level contains 260 instances.

Variation across groups. Mixture-SFT's five-run mean pass@1 ranges from 20.0% on Routing to 62.0% on Logical. Scheduling (30.0%) is also below its overall mean of 38.8%. Network averages 40.9%, but its SD of 18.5 percentage points indicates substantial variation across inference runs. Mean pass@1 decreases from 53.4% on easy instances to 38.5% on medium and 24.4% on hard instances, a 29.0-percentage-point easy–hard gap. These breakdowns describe performance on the benchmark's groups; they do not isolate the effects of family or difficulty.

Comparison with SG-DPO. Relative to Mixture-SFT, SG-DPO has higher five-run mean pass@1 in all six families and all three difficulty groups. On hard instances, the mean increases from 24.4% to 34.1%. Pass@4 is also higher in all nine groups, rising from 41.9% to 47.3% on hard instances. These results assess the complete DPO procedure, not the separate contributions of its components (§F.6).

Table 21: Pass@1 (%) by MM-OptBench family and difficulty. Subscripts give five-run sample SD; cells without an SD give first-run accuracy. Evaluation protocol: §B.2. –: pending. Coverage and provenance: §G.

		Optimization family						Difficulty			
	Model	Network	Location	Scheduling	Planning	Routing	Logical	Easy	Medium	Hard	Overall
	N	150	150	120	120	120	120	260	260	260	780
General MLLMs	Claude Sonnet 4.6	$10.0_{\pm 5.8}$	$20.7_{\pm 1.5}$	$57.5_{\pm 10.8}$	$39.2_{\pm 12.0}$	$15.8_{\pm 5.4}$	$6.7_{\pm 4.8}$	$37.3_{\pm 9.3}$	$21.5_{\pm 3.2}$	$13.8_{\pm 5.0}$	$24.2_{\pm 4.5}$
	Gemini 3.1 Pro	$62.7_{\pm 6.4}$	$56.0_{\pm 5.5}$	$59.2_{\pm 12.6}$	$56.7_{\pm 9.1}$	$50.0_{\pm 5.9}$	$60.8_{\pm 9.1}$	$73.1_{\pm 8.8}$	$56.9_{\pm 2.9}$	$43.1_{\pm 5.2}$	$57.7_{\pm 5.2}$
	GPT-5.5	$33.9_{\pm 2.1}$	$69.2_{\pm 2.5}$	$81.3_{\pm 1.8}$	$77.3_{\pm 3.0}$	$41.2_{\pm 3.7}$	$83.2_{\pm 2.5}$	$71.8_{\pm 1.5}$	$63.2_{\pm 2.0}$	$55.1_{\pm 1.2}$	$63.4_{\pm 0.5}$
	Kimi-K2.5	$45.3_{\pm 6.1}$	$54.0_{\pm 4.3}$	$70.8_{\pm 9.8}$	$62.5_{\pm 10.2}$	$35.8_{\pm 5.6}$	$55.0_{\pm 5.4}$	$69.2_{\pm 4.9}$	$57.7_{\pm 4.3}$	$33.8_{\pm 2.2}$	$53.6_{\pm 2.8}$
	Qwen3.6-Plus	$11.6_{\pm 1.6}$	$15.5_{\pm 0.6}$	$22.2_{\pm 4.1}$	$18.3_{\pm 7.0}$	$7.3_{\pm 1.2}$	$24.0_{\pm 5.7}$	$23.2_{\pm 2.5}$	$16.8_{\pm 2.5}$	$8.8_{\pm 2.3}$	$16.3_{\pm 1.4}$
	GLM-4.5V	$12.1_{\pm 2.6}$	$9.2_{\pm 2.6}$	$8.3_{\pm 2.6}$	$11.2_{\pm 2.1}$	$6.8_{\pm 3.2}$	$19.3_{\pm 1.6}$	$18.5_{\pm 2.4}$	$10.9_{\pm 2.1}$	$3.9_{\pm 0.8}$	$11.1_{\pm 1.1}$
Math MLLMs	MathCoder-VL-8B	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$
	MM-Eureka	$0.0_{\pm 0.0}$	$0.7_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.2_{\pm 0.4}$	$0.2_{\pm 0.4}$	$1.2_{\pm 0.5}$	$1.1_{\pm 0.4}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.4_{\pm 0.1}$
	MM-PRM	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$
Qwen3.5 Fine-tuned Models	Qwen3.5-9B	$13.7_{\pm 4.5}$	$14.5_{\pm 5.6}$	$8.8_{\pm 3.8}$	$5.2_{\pm 2.5}$	$3.2_{\pm 2.1}$	$24.3_{\pm 4.8}$	$17.6_{\pm 3.7}$	$10.5_{\pm 3.7}$	$7.4_{\pm 3.0}$	$11.8_{\pm 3.4}$
	Direct-SFT	$53.6_{\pm 2.2}$	$43.7_{\pm 2.2}$	$18.0_{\pm 3.2}$	$38.5_{\pm 4.1}$	$23.8_{\pm 2.3}$	$63.5_{\pm 3.9}$	$49.8_{\pm 1.0}$	$39.6_{\pm 1.5}$	$33.1_{\pm 2.9}$	$40.8_{\pm 1.3}$
	Compact-SFT	$20.1_{\pm 2.6}$	$42.8_{\pm 1.1}$	$6.3_{\pm 2.9}$	$30.3_{\pm 4.9}$	$9.3_{\pm 4.0}$	$68.2_{\pm 1.8}$	$40.0_{\pm 2.8}$	$28.5_{\pm 1.6}$	$20.5_{\pm 1.7}$	$29.7_{\pm 1.2}$
	Detailed-SFT	$35.7_{\pm 1.5}$	$40.1_{\pm 2.6}$	$10.8_{\pm 2.6}$	$11.5_{\pm 3.8}$	$10.5_{\pm 6.3}$	$63.0_{\pm 1.6}$	$37.4_{\pm 1.3}$	$26.7_{\pm 0.9}$	$23.9_{\pm 2.9}$	$29.3_{\pm 1.1}$
	Mixture-SFT	$40.9_{\pm 18.5}$	$39.3_{\pm 5.2}$	$30.0_{\pm 4.0}$	$39.7_{\pm 3.8}$	$20.0_{\pm 1.9}$	$62.0_{\pm 5.6}$	$53.4_{\pm 2.9}$	$38.5_{\pm 5.6}$	$24.4_{\pm 10.9}$	$38.8_{\pm 6.5}$
	SG-DPO	$53.9_{\pm 2.0}$	$43.1_{\pm 1.5}$	$33.8_{\pm 4.5}$	$43.3_{\pm 2.3}$	$22.8_{\pm 2.0}$	$67.0_{\pm 0.5}$	$56.1_{\pm 1.5}$	$42.8_{\pm 1.8}$	$34.1_{\pm 1.5}$	$44.3_{\pm 0.8}$

Table 22: Pass@4 (%) by MM-OptBench family and difficulty. Evaluation protocol: §B.2. –: pending. Coverage and provenance: §G.

		Optimization family						Difficulty			
	Model	Network	Location	Scheduling	Planning	Routing	Logical	Easy	Medium	Hard	Overall
	N	150	150	120	120	120	120	260	260	260	780
General MLLMs	Claude Sonnet 4.6	20.0	33.3	87.5	62.5	25.0	16.7	53.8	38.5	26.9	39.7
	Gemini 3.1 Pro	83.3	70.0	83.3	87.5	66.7	70.8	90.4	76.9	63.5	76.9
	GPT-5.5	54.7	73.3	90.0	90.0	55.8	97.5	85.0	75.8	66.9	75.9
	Kimi-K2.5	66.7	63.3	87.5	79.2	41.7	66.7	82.7	71.2	48.1	67.3
	Qwen3.6-Plus	28.0	41.3	50.0	42.5	19.2	57.5	53.8	38.5	25.8	39.4
	GLM-4.5V	30.0	27.3	23.3	28.3	22.5	40.0	48.8	26.9	10.0	28.6
Math MLLMs	MathCoder-VL-8B	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	MM-Eureka	0.0	0.7	0.0	0.0	0.8	1.7	1.5	0.0	0.0	0.5
	MM-PRM	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Qwen3.5 Fine-tuned Models	Qwen3.5-9B	38.7	37.3	26.7	20.0	8.3	54.2	44.6	27.3	22.3	31.4
	Direct-SFT	67.3	50.7	42.5	54.2	29.2	78.3	65.0	51.9	45.4	54.1
	Compact-SFT	37.3	52.7	19.2	51.7	23.3	87.5	56.9	45.0	33.8	45.3
	Detailed-SFT	57.3	57.3	28.3	30.8	19.2	77.5	57.3	43.5	37.3	46.0
	Mixture-SFT	61.3	46.0	57.5	49.2	30.8	76.7	65.8	53.1	41.9	53.6
	SG-DPO	62.0	48.0	58.3	51.7	38.3	82.5	67.7	55.0	47.3	56.7

G.4 LLM4OR-VL Results by Source

Table 23 breaks down LLM4OR-VL performance by its eight source collections. C-OR and I-OR denote ComplexOR and IndustryOR; MAMO-C and MAMO-E denote MAMO Complex LP and Easy LP. OptMATH denotes OptMATH-Bench, and ReSoc denotes ReSocratic. Column headers give the numbers of retained instances, which range from 18 to 192 per source. The Overall column pools all 878 instances, so larger source groups carry more weight; it is not an equal average of the eight source scores.

Variation across sources. Mixture-SFT has higher five-run mean pass@1 than Qwen3.5-9B on all eight sources. The largest difference is on MAMO Easy LP (85.9% versus 32.3%), while the difference on OptMATH-Bench is much smaller (19.9% versus 18.6%). Relative to Mixture-SFT, SG-DPO has higher mean pass@1 on seven sources; its mean on MAMO Complex LP is slightly lower (47.5% versus 48.1%). These comparisons follow the reporting scope in §G.

OptMATH-Bench is Mixture-SFT's lowest-scoring source, in contrast to its strong performance on OptMATH-derived MLLM4OR-Test instances. The latter are reconstructed from OptMATH-Train, whereas the LLM4OR-VL group is reconstructed from OptMATH-Bench (§C.2, §D.3). The two groups therefore use different source collections and reconstructions; their shared name does not imply comparable evaluation difficulty.

Table 23: Source-level pass@1 (%) on LLM4OR-VL. Subscripts give five-run sample SD; cells without an SD give first-run accuracy. Evaluation protocol: §B.2. –: pending. Coverage and provenance: §G.

	Model	C-OR	I-OR	MAMO-C	MAMO-E	NL4Opt	NLP4LP	OptMATH	ReSoc	Overall
	N	18	42	106	109	67	178	166	192	878
General MLLMs	Claude Sonnet 4.6	$45.0_{\pm 11.2}$	$5.0_{\pm 11.2}$	$52.4_{\pm 9.5}$	$73.6_{\pm 12.6}$	$46.2_{\pm 12.2}$	$68.9_{\pm 7.7}$	$53.9_{\pm 6.9}$	$70.8_{\pm 9.9}$	$59.8_{\pm 5.0}$
	Gemini 3.1 Pro	$95.0_{\pm 11.2}$	$75.0_{\pm 15.3}$	$75.2_{\pm 4.0}$	$77.3_{\pm 8.5}$	$44.6_{\pm 12.6}$	$83.3_{\pm 5.2}$	$70.9_{\pm 4.6}$	$87.2_{\pm 3.1}$	$77.0_{\pm 1.5}$
	GPT-5.5	$84.4_{\pm 4.6}$	$75.7_{\pm 4.3}$	$94.9_{\pm 3.0}$	$93.6_{\pm 2.2}$	$69.6_{\pm 3.4}$	$88.0_{\pm 2.8}$	$84.2_{\pm 1.4}$	$86.5_{\pm 1.8}$	$86.4_{\pm 0.4}$
	Kimi-K2.5	$75.0_{\pm 0.0}$	$72.5_{\pm 5.6}$	$90.5_{\pm 3.4}$	$87.3_{\pm 3.8}$	$67.7_{\pm 8.4}$	$78.9_{\pm 6.4}$	$72.7_{\pm 7.7}$	$93.3_{\pm 3.4}$	$82.1_{\pm 2.8}$
	Qwen3.6-Plus	$15.6_{\pm 7.2}$	$15.2_{\pm 6.9}$	$28.5_{\pm 7.0}$	$39.8_{\pm 6.3}$	$11.6_{\pm 4.9}$	$21.0_{\pm 1.9}$	$18.1_{\pm 2.8}$	$23.5_{\pm 3.0}$	$23.1_{\pm 2.9}$
	GLM-4.5V	$45.6_{\pm 7.2}$	$30.0_{\pm 10.5}$	$44.2_{\pm 2.9}$	$69.0_{\pm 5.9}$	$49.3_{\pm 5.0}$	$51.3_{\pm 3.5}$	$14.0_{\pm 4.2}$	$61.0_{\pm 5.0}$	$46.4_{\pm 1.9}$
Math MLLMs	MathCoder-VL-8B	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$
	MM-Eureka	$24.4_{\pm 3.0}$	$24.8_{\pm 2.1}$	$26.4_{\pm 0.7}$	$73.8_{\pm 1.0}$	$18.5_{\pm 1.3}$	$35.3_{\pm 1.7}$	$1.1_{\pm 0.5}$	$49.3_{\pm 1.6}$	$33.6_{\pm 0.6}$
	MM-PRM	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$
Qwen3.5 Fine-tuned Models	Qwen3.5-9B	$34.4_{\pm 7.2}$	$21.4_{\pm 3.4}$	$27.2_{\pm 2.5}$	$32.3_{\pm 4.7}$	$26.3_{\pm 5.9}$	$29.9_{\pm 5.6}$	$18.6_{\pm 3.3}$	$34.0_{\pm 5.3}$	$28.0_{\pm 3.4}$
	Direct-SFT	$51.1_{\pm 13.3}$	$25.2_{\pm 4.9}$	$42.3_{\pm 6.1}$	$76.9_{\pm 3.3}$	$31.6_{\pm 1.9}$	$44.8_{\pm 1.6}$	$7.8_{\pm 1.2}$	$50.6_{\pm 2.4}$	$41.0_{\pm 0.9}$
	Compact-SFT	$50.0_{\pm 9.6}$	$24.8_{\pm 1.3}$	$47.5_{\pm 3.2}$	$60.9_{\pm 5.0}$	$38.5_{\pm 8.9}$	$57.9_{\pm 3.7}$	$10.5_{\pm 2.6}$	$57.9_{\pm 1.8}$	$44.8_{\pm 0.9}$
	Detailed-SFT	$38.9_{\pm 8.8}$	$21.0_{\pm 5.7}$	$27.0_{\pm 4.3}$	$53.4_{\pm 2.9}$	$22.4_{\pm 1.5}$	$33.7_{\pm 4.5}$	$7.6_{\pm 1.2}$	$34.6_{\pm 2.7}$	$29.2_{\pm 1.7}$
	Mixture-SFT	$45.6_{\pm 12.0}$	$27.6_{\pm 6.2}$	$48.1_{\pm 5.4}$	$85.9_{\pm 5.6}$	$35.8_{\pm 8.6}$	$48.3_{\pm 8.2}$	$19.9_{\pm 2.7}$	$51.2_{\pm 6.4}$	$46.2_{\pm 5.7}$
	SG-DPO	$50.0_{\pm 6.8}$	$32.4_{\pm 4.3}$	$47.5_{\pm 2.7}$	$91.0_{\pm 2.0}$	$40.6_{\pm 7.0}$	$53.6_{\pm 3.3}$	$21.8_{\pm 3.6}$	$53.9_{\pm 3.4}$	$49.5_{\pm 2.8}$

G.5 MLLM4OR-Test Results by Source

Table 24 reports results for the two sources in MLLM4OR-Test: 1,254 OptMATH-derived and 211 OR-Instruct-derived instances. The generator branch contributes no test instances. Overall pools all 1,465 instances rather than averaging the two source accuracies equally. The comparisons below use five-run mean pass@1, consistent with Table 1.

Mixture-SFT versus Direct-SFT. Mixture-SFT has higher mean pass@1 in both source groups: 94.3% versus 82.0% on OptMATH-derived instances, and 71.0% versus 65.1% on OR-Instruct-derived instances. However, its two source means still differ by 23.3 percentage points. OptMATH-derived instances account for 85.6% of the test set, so the overall mean of 91.0% is closer to the OptMATH result and does not convey the lower accuracy on OR-Instruct.

SG-DPO versus Mixture-SFT. SG-DPO has lower mean pass@1 in both groups: 94.0% on OptMATH-derived instances and 67.2% on OR-Instruct-derived instances, with the larger decrease on the latter. Its overall mean is 90.1%, compared with Mixture-SFT's 91.0%. Thus its higher three-suite aggregate reflects gains on the external benchmarks, not higher mean accuracy on MLLM4OR-Test.

Table 24: Source-level pass@1 (%) on MLLM4OR-Test. Subscripts give five-run sample SD; cells without an SD give first-run accuracy. Evaluation protocol: §B.2. –: pending. Coverage and provenance: §G.

	Model	OptMATH-derived	OR-Instruct-derived	Overall
	N	1,254	211	1,465
General MLLMs	Claude Sonnet 4.6	$49.8_{\pm 6.5}$	$89.0_{\pm 8.2}$	$55.5_{\pm 6.5}$
	Gemini 3.1 Pro	$82.1_{\pm 3.5}$	$84.5_{\pm 6.0}$	$82.5_{\pm 3.2}$
	GPT-5.5	$84.6_{\pm 0.3}$	$89.0_{\pm 1.6}$	$85.3_{\pm 0.4}$
	Kimi-K2.5	$80.9_{\pm 0.8}$	$85.6_{\pm 4.5}$	$81.6_{\pm 1.3}$
	Qwen3.6-Plus	$14.3_{\pm 0.4}$	$28.1_{\pm 5.7}$	$16.3_{\pm 1.1}$
	GLM-4.5V	$36.3_{\pm 1.1}$	$46.8_{\pm 5.5}$	$37.9_{\pm 0.9}$
Math MLLMs	MathCoder-VL-8B	$0.0_{\pm 0.0}$	$0.1_{\pm 0.2}$	$0.0_{\pm 0.0}$
	MM-Eureka	$9.6_{\pm 0.4}$	$49.8_{\pm 1.0}$	$15.4_{\pm 0.4}$
	MM-PRM	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$
Qwen3.5 Fine-tuned Models	Qwen3.5-9B	$25.3_{\pm 3.7}$	$31.3_{\pm 7.1}$	$26.2_{\pm 4.2}$
	Direct-SFT	$82.0_{\pm 0.4}$	$65.1_{\pm 2.3}$	$79.5_{\pm 0.4}$
	Compact-SFT	$54.3_{\pm 0.8}$	$53.0_{\pm 4.0}$	$54.1_{\pm 1.0}$
	Detailed-SFT	$42.6_{\pm 1.2}$	$48.2_{\pm 3.0}$	$43.4_{\pm 0.9}$
	Mixture-SFT	$94.3_{\pm 0.2}$	$71.0_{\pm 2.8}$	$91.0_{\pm 0.4}$
	SG-DPO	$94.0_{\pm 0.3}$	$67.2_{\pm 4.0}$	$90.1_{\pm 0.8}$

G.6 Failure Types and Paired Changes

Analysis population. Following the normalized failure-bar presentation in PEARL (Lu et al., 2026), we compare the reported Qwen3.5-9B backbone with Direct-SFT, Compact-SFT, Detailed-SFT, Mixture-SFT, and SG-DPO using saved greedy responses. Of the 3,123 test instances, 21 have no backbone response because of service failures; we exclude these same IDs from all six models. The resulting common panel contains 3,102 instances: 1,452 from MLLM4OR-Test, 877 from LLM4OR-VL, and 773 from MM-OptBench, yielding 18,612 model–instance outcomes. Terminal model outputs that exhaust the generation budget remain failures, rather than being excluded as service errors. This is a greedy-only analysis, separate from the five-run means and pass@4 reported elsewhere; published scores and their denominators are unchanged.

Failure attribution. We classify failures using the saved answers and their execution records, without regenerating or repairing answers.

Error categories. Each failed answer receives one label, prioritizing output sections, syntax, source-format and safety checks, execution, output validity, and reference agreement in that order. These descriptive labels refine the evaluation outcomes; they do not replace the F0–F4 labels used for preference construction.

- **Format/extraction:** missing required output sections, an invalid required result marker or result-payload format, or missing or ambiguous required objective output. Truncated outputs without the required sections are included here.
- **Code/runtime:** invalid Python syntax, a static-safety or solver-backend rejection, or a nonzero program exit before a usable result is obtained.
- **Timeout:** an otherwise eligible program exceeds the 600-second execution limit.
- **Infeasible/solver:** a normally terminating program reports a solver status that does not satisfy the reference requirement, or the solver provides no required feasible solution or objective.
- **Objective mismatch:** a usable objective is obtained, but differs from the reference beyond the scoring tolerance.
- **Other:** a returned feasibility witness fails the instance-specific checks. Missing evidence is not assigned to this category.

Output-section, syntax, and static checks precede execution; execution must finish normally before its result is parsed and compared with the reference. Consequently, a program that crashes after encountering infeasibility is labeled Code/runtime, rather than Infeasible/solver. The labels identify observed failure stages, not whether the original mistake arose from visual interpretation, mathematical formulation, or implementation. Infrastructure failures would remain unresolved rather than becoming model-error labels.

Failure composition across models. Code/runtime errors form the largest failure category for all five fine-tuned models, accounting for 63.9%, 70.3%, and 74.8% of failures in Direct-SFT, Compact-SFT, and Detailed-SFT. Mixture-SFT and SG-DPO have similar profiles: 486 and 492 code/runtime errors, and 389 and 380 objective mismatches, respectively (Table 25). Their objective mismatches account for 40.5% and 39.6% of failures. These categories distinguish unsuccessful execution from disagreement with the reference objective; they do not establish whether an error originates in visual interpretation, formulation, or implementation. Neither a category's share nor its count alone isolates a change in the model's underlying reasoning ability; the first-failure convention masks errors beyond the earliest failed check.

Table 25: Greedy failure counts on the common 3,102-instance panel. Categories are mutually exclusive and sum to the final column.

Model	Code/ runtime	Objective mismatch	Infeasible/ solver	Format/ extraction	Timeout	Other	Failures
Qwen3.5-9B	481	151	80	1,464	4	1	2,181
Direct-SFT	801	335	76	24	0	17	1,253
Compact-SFT	1,163	365	65	59	1	2	1,655
Detailed-SFT	1,456	261	115	107	1	7	1,947
Mixture-SFT	486	389	47	32	4	2	960
SG-DPO	492	380	46	34	5	2	959

Paired changes after DPO. For the complementary Mixture-SFT–SG-DPO comparison, both models have responses on all 3,123 test instances, so we retain that full panel and match by test suite, instance ID, and reference record. SG-DPO solves 38 instances that Mixture-SFT fails. Their original Mixture-SFT errors comprise 21 code/runtime errors, 13 objective mismatches, two solver-status failures, one format failure, and one invalid feasibility witness. Conversely, 37 previously successful instances fail after DPO; Table 26 reports both directions. The net gain is therefore one correct answer, not 38. These greedy-only transition counts should not be substituted for the five-run mean comparison in §4.3.

Table 26: Paired greedy outcomes for Mixture-SFT and SG-DPO. Gains are failure-to-success transitions; losses are success-to-failure transitions. The net column equals gains minus losses.

Test suite	Instances	Gains	Losses	Net
MLLM4OR-Test	1,465	1	2	−1
LLM4OR-VL	878	12	19	−7
MM-OptBench	780	25	16	+9
All three	3,123	38	37	+1

G.7 Mathematical-Baseline Evaluation Scope

Evaluation scope. We evaluate MathCoder-VL-8B, MM-Eureka, and MM-PRM as generators of optimization-modeling programs. The published MM-PRM method scores reasoning paths for Best-of-N selection (Du et al., 2025); here, the evaluated checkpoint generates answers directly, without reward-based candidate selection. Its scores therefore do not evaluate the published process-reward pipeline. The available records do not establish its exact checkpoint revision.

Execution-based evaluation. Primary scores use the generated programs without modifying their statements. Correctness is checked against the reference for the same instance using the execution-based criteria in §B.2. Repaired programs are excluded from primary scores.

Comparison limits. Image processing differs across these baselines, and MM-Eureka's sampled-run records do not establish a common effective image resolution with its greedy runs. Their reported means are therefore not comparisons under identical visual-processing settings. MathCoder-VL-8B's sampled zero scores combine local MLLM4OR-Test execution evidence with author-confirmed aggregates for untransferred samples; MM-PRM's sampled zero scores are author-confirmed aggregates.