VISION-LANGUAGE MODELS AS END-TO-END COMBINATORIAL OPTIMIZATION SOLVERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have shown promise for automating combinatorial optimization (CO), yet existing approaches rely on textual representations of problem instances. Many real-world CO problems are naturally presented in visual forms, and text-based encodings may lead to spatial information loss and limit spatial reasoning capabilities. In this paper, we introduce a novel framework that empowers vision-language models (VLMs) to serve as end-to-end CO solvers by directly processing visual problem instances. We design visual-attributed instances that encode problem structures and basic features into images, and develop self-evolutionary visual prompt optimization to automatically discover effective visual designs. Our two-stage training strategy first uses supervised fine-tuning (SFT) to learn solution patterns from domain-specific solvers, followed by reinforcement learning to reduce constraint violations and improve solution quality. Additionally, we introduce inference-time visual augmentation that exploits structure-preserving transformations to generate diverse candidate solutions. Experiments on six NP-hard CO problems show that our method achieves 100% feasibility, with optimality gaps ranging from 1.78% to 10.93%. It improves average feasibility over advanced reasoning models, including GPT-6-Astra and Gemini-2.5-Pro, by 26.5% and 44.3%, respectively, while also outperforming specialized heuristics.

1 INTRODUCTION

Combinatorial optimization (CO) underlies many critical decision-making tasks in transportation, logistics, and manufacturing (Da Ros et al., 2026). Traditional solvers typically rely on exact optimization methods or carefully designed problem-specific heuristics (Peres & Castelli, 2021; Yang, 2010). The former can provide optimality guarantees but often scale poorly with increasing problem size, whereas the latter improve computational efficiency at the cost of substantial manual design effort. Neural combinatorial optimization (NCO) (Kwon et al., 2020; Wang et al., 2024) offers an alternative by learning solution policies end to end. However, existing NCO methods remain largely specialized to individual problem classes and require extensive task-specific training.

Large language models (LLMs) have recently been explored for CO through automated heuristic design (Ye et al., 2024), tool-assisted optimization (Jiang et al., 2025), and direct solution generation (Jiang et al., 2026). These approaches, however, primarily operate on textual or symbolic representations of problem instances. In contrast, many real-world CO problems admit natural and intuitive visual representations, such as spatial maps for routing, scheduling diagrams for production planning, and graph structures for resource allocation. Meanwhile, vision-language models (VLMs), including GPT-4o and Gemini-2.5-Pro, have demonstrated strong visual understanding and multimodal reasoning capabilities (Masry et al., 2022; Xu et al., 2026). These recent advances motivate the development of VLM-based CO solvers that directly operate on visual problem representations without explicit textual instance encoding.

Recent studies have begun to explore VLMs for planning and optimization. Existing work applies VLMs to navigation (Zhang et al., 2025), embodied decision-making (Ma et al., 2026), and visually grounded planning (Xu et al., 2026), while early efforts in CO use VLMs to identify promising subproblems for specialized solvers (Yin et al., 2026) or construct solutions through multi-agent visual reasoning (Elhenawy et al., 2024). These studies demonstrate the potential of visual reasoning for optimization, but they are typically restricted to specific problem classes or rely on iterative

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

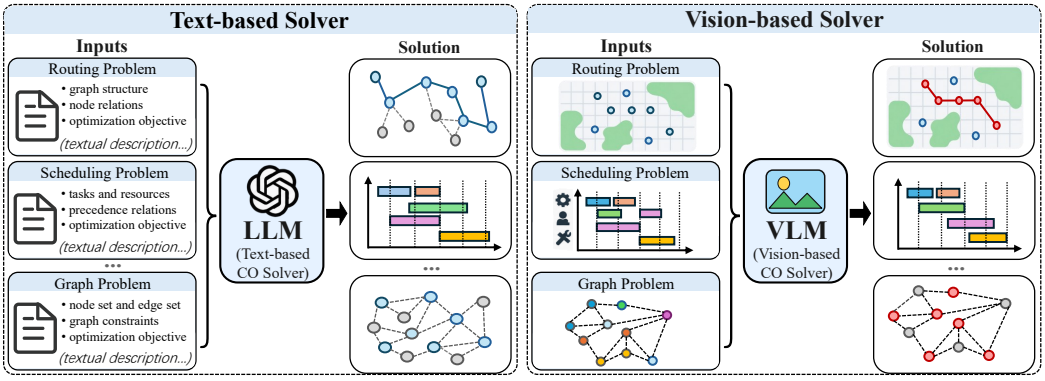


Figure 1: Illustration of vision-language models for combinatorial optimization.

prompting, external search, or specialized solvers. Whether a VLM can be trained as a general end-to-end solver that directly maps visual instances to high-quality solutions across diverse CO problems remains largely unexplored.

In this work, we propose a framework for training VLMs as end-to-end solvers for diverse CO problems from visual inputs. We consider three major CO problem families: routing (TSP, CVRP), graph optimization (MIS, MVC), and scheduling (JSSP, PFSP). We first design tailored visual representations for each family, and further optimize these representations through LLM-guided evolutionary search. We then train the VLM in two stages: supervised fine-tuning learns solution generation by imitating domain-specific solvers, followed by reinforcement learning with verifiable rewards to improve constraint satisfaction and solution quality. At inference time, semantics-preserving visual augmentation further increases solution diversity.

Our contributions are summarized as follows. **(1)** We introduce an end-to-end visual CO solver that directly maps visual problem instances to solutions, and construct a 3-million-image dataset spanning six CO problems across three families using a self-evolutionary visual prompt optimization method. **(2)** We develop a two-stage training framework combining supervised fine-tuning and reinforcement learning, together with inference-time multi-view augmentation to improve solution quality and diversity. **(3)** We demonstrate that our 7B VLM outperforms advanced reasoning models such as GPT-6-Astra and Gemini-2.5-Pro, as well as specialized heuristics across all six problems without problem-specific architectural modifications.

2 Related Work

Combinatorial Optimization. Many combinatorial optimization problems are NP-hard, making exact optimization intractable as problem size grows Papadimitriou & Steiglitz (1998), which motivates research on problem-specific heuristics and metaheuristics Rosenkrantz et al. (1977). Though effective, such methods rely on substantial domain expertise and transfer poorly across problem classes. Neural combinatorial optimization reduces manual algorithm design through end-to-end learning Vinyals et al. (2015); Kool et al. (2019), but most methods remain specialized to individual problem families and require dedicated training. Recently, LLMs have been explored as end-to-end solvers Jiang et al. (2026); Yang et al. (2024), program synthesizers with retrieval-augmented generation Romera-Paredes et al. (2024); Jiang et al. (2025); Tao et al. (2025), and hyper-heuristic generators Ye et al. (2024). However, existing approaches predominantly operate on textual or symbolic representations, leaving visually represented CO problems largely unexplored.

VLMs for Optimization. Vision-language models, such as GPT-4V, Gemini, and LLaVA, demonstrate remarkable capabilities in understanding visual content. VLMs can process complex visual information including charts Liu et al. (2024b), diagrams Li et al. (2025), and structured data Li et al. (2023a), suggesting potential for visually-encoded optimization problems. However, existing applications focus primarily on perception tasks Antol et al. (2015); Li et al. (2023b) or mathematical reasoning Lu et al. (2024); Zou et al. (2025). VLM-based reasoning has adopted structured prompting techniques, particularly chain of thought (CoT) Wei et al. (2022), which emphasizes stepwise reasoning. Recent work has explored pure visual reasoning without language mediation for

spatial planning Xu et al. (2026) and vision-language graph reasoning Wei et al. (2024). However, step-by-step reasoning becomes impractical for CO problems, especially large-scale ones. Recent efforts have explored VLMs for CO through visual evolutionary optimization Zhao & Cheong (2025), graph-structured problem solving Wei et al. (2024), implicit optimization Mirza et al. (2024), geometry-aware routing agents Zhang et al. (2026), multi-agent visual reasoning for TSP Elhenawy et al. (2024), and guided TSP frameworks Yin et al. (2026). In contrast, we train VLMs to directly generate CO solutions from optimized visual representations by imitating domain-specific solvers, without explicit reasoning chains or iterative search.

3 Problem Statement

We investigate VLMs as end-to-end CO solvers that directly map visual problem instances to complete solutions, without explicit textual instance encodings. A CO problem class Π (e.g., routing, scheduling, or graph optimization) defines an instance space whose elements $q \in \Pi$ are individual problem instances. Each instance is characterized by a solution space $\mathcal{S}_q$, an objective function $\mathcal{O}_q : \mathcal{S}_q \to \mathbb{R}$, and a set of constraints $\mathcal{C}_q = \{c_{q,1}, \ldots, c_{q,m_q}\}$. Each constraint $c_{q,j} : \mathcal{S}_q \to \{0, 1\}$ indicates whether a candidate solution satisfies the j-th requirement. The feasible set is therefore given by

$$\mathcal{F}_q = \{x \in \mathcal{S}_q \mid c_{q,j}(x) = 1,\ \forall j = 1, \ldots, m_q\}. \tag{1}$$

We state the formulation for minimization problems; maximization problems (e.g., MIS) are handled by negating the objective. The optimal solution x_q^* is defined in Eq. (2):

$$x_q^* = \arg\min_{x \in \mathcal{F}_q} \mathcal{O}_q(x). \tag{2}$$

To formulate visual CO solving, let $\rho : \Pi \to \mathcal{I}$ denote a rendering function that transforms an instance q into a visual representation $I_q = \rho(q)$. We denote the VLM policy by $\pi_\theta : \mathcal{I} \times \mathcal{L} \to \mathcal{L}$, where $\mathcal{I}$ and $\mathcal{L}$ denote the image and language spaces, respectively. For each problem class, a task-level instruction $u_\Pi \in \mathcal{L}$ specifies the optimization objective, constraints, and required output format, while the instance-specific information is provided only through I_q. The generated response is parsed into a valid solution representation by $\delta_q : \mathcal{L} \to \mathcal{S}_q$. The resulting end-to-end solving process, which composes rendering, inference, and parsing (Eq. (3)), is

$$\hat{x}_q = \delta_q(\pi_\theta(\rho(q), u_\Pi)). \tag{3}$$

The goal is to learn π_θ such that $\hat{x}_q$ satisfies the problem constraints while achieving an objective value close to x_q^*. Given a benchmark set $\mathcal{B}$ of instances, we measure feasibility as $\mathrm{Feas}(\mathcal{B}) = \frac{1}{|\mathcal{B}|}\sum_{q \in \mathcal{B}} \mathbb{I}[\hat{x}_q \in \mathcal{F}_q]$ and solution quality using the optimality gap $\mathrm{Gap}(\mathcal{B}) = \frac{1}{|\mathcal{B}|}\sum_{q \in \mathcal{B}} \frac{\mathcal{O}_q(\hat{x}_q) - \mathcal{O}_q(x_q^*)}{|\mathcal{O}_q(x_q^*)|}$, where instances with infeasible solutions contribute a gap of ∞. Since solutions to several of our problems (routing, scheduling) are sequences rather than sets, we treat $\mathcal{S}_q$ as a space of ordered sequences throughout, so that membership in $\mathcal{F}_q$ is well defined.

4 Methodology

4.1 Visual Instance Construction

Figure 2 illustrates the overall framework. To train VLMs for CO, we construct visual problem solution pairs by rendering CO instances as images and pairing them with high quality reference solutions. Unlike text based LLM solvers (Jiang et al., 2026) that serialize instances into structured representations, our approach directly encodes problem structures visually. Reference solutions are generated using LKH (Helsgaun, 2017) for TSP and CVRP, Q learning based iterated greedy (QIG) for PFSP and JSSP, and Gurobi (Muley, 2021) for MIS and MVC.

Visual Instance Rendering. Tailored rendering approaches are designed for three problem families with specific visual encoding parameters:

- **Routing (V-TSP/V-CVRP).** Instances with $n \sim \mathcal{U}[10, 100] \cap \mathbb{Z}$ nodes are rendered on a 25×25 coordinate grid as 2-D scatter plots with DPI=100. Coordinates are mapped to grid positions with

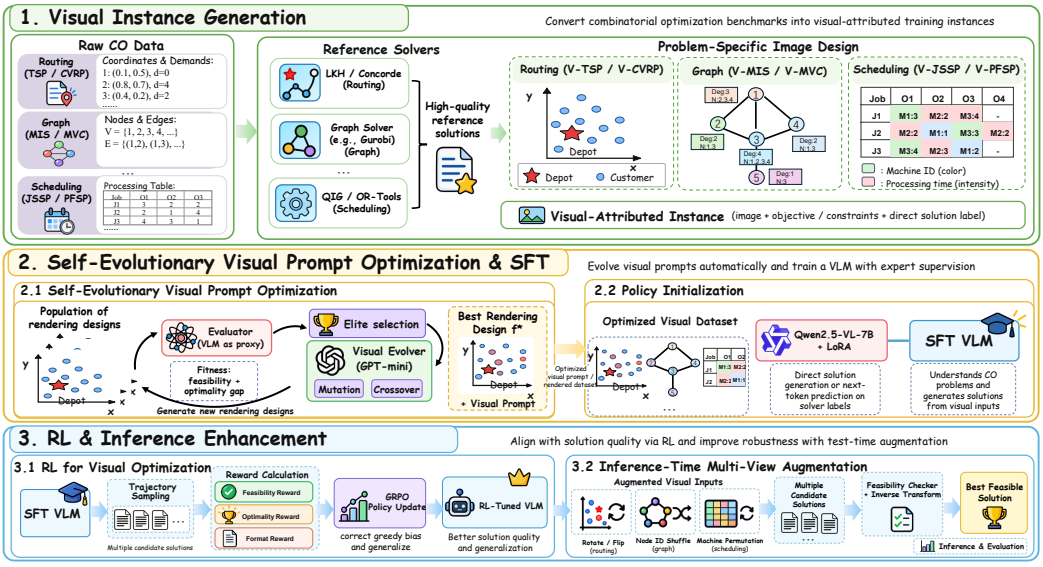


Figure 2: Overview of the our proposed VLM-CO framework.

automatic jitter applied to overlapping nodes. The depot (node 0) is marked with a star symbol, while customer nodes appear as labeled circles.

- **Graph optimization (V-MIS/V-MVC).** Graphs with $n \sim \mathcal{U}[10, 100] \cap \mathbb{Z}$ nodes are rendered within a fixed invisible grid. Each vertex is displayed as a box showing node id, degree, and neighbor ids. Node positions are constrained to grid cells to prevent visual clutter.
- **Scheduling (V-JSSP/V-PFSP).** Instances with $J, M \sim \mathcal{U}[5, 20] \cap \mathbb{Z}$ jobs and machines are rendered as tile-based layouts. For JSSP, the left column is fixed to job indices, with arrows indicating precedence constraints between operations. For PFSP, the left column is fixed to machine indices.

Visual-Attributed Instance. A Visual-Attributed Instance (VAI) consists of (1) **Input**: a rendered problem image paired with a textual description of the optimization objectives and constraints, and (2) **Output**: a structured textual solution. For example, a TSP solution reads: "Routes: [0, 2, 4, 6, 1, 5, 3, 0], Objective: 6355.86".

4.2 Self-Evolutionary Visual Prompt Optimization

While Section 4.1 provides initial rendering schemes, visual encoding choices (e.g., heuristic-based coloring, node size and shape, image resolution) significantly impact VLM learning efficiency and solution quality. We formulate visual prompt design as a black-box optimization problem and employ LLM-guided evolutionary search to systematically optimize these design choices (Liu et al., 2024a). Implementation details of the visual evolution process are provided in Appendix C.

Let $\mathcal{D} = \{(q_i, s_i)\}_{i=1}^{N}$ denote a dataset of N combinatorial optimization problem instances, where q_i represents the problem input and s_i the optimal solution. A rendering function $\rho : \Pi \to \mathcal{I}$ maps a problem instance q to an image representation $I \in \mathcal{I}$. Given a base vision-language model π_{θ_0} and a rendering function ρ, the task-specific model π_{θ_ρ} is obtained by supervised fine-tuning on the rendered dataset $\mathcal{D}_\rho = \{(\rho(q_i), s_i)\}_{i=1}^{N}$. The optimization objective is stated in Eq. (4):

$$\rho^* = \arg\min_{\rho} \mathcal{J}(\pi_{\theta_\rho}; \mathcal{D}) \tag{4}$$

Visual Evolver. We maintain a population $\mathcal{P}_t = \{\rho_1, \ldots, \rho_K\}$ of K rendering functions at generation t, where each $\rho_i \in \mathcal{R}$ is represented as executable Python code parameterizing visual design choices (e.g., spatial coloring). At each generation, we select the elite individual $\rho_{\text{best}} = \arg\max_{\rho_i \in \mathcal{P}_t} \Phi(\rho_i)$ with the highest fitness $\Phi : \mathcal{R} \to \mathbb{R}$, and generate offspring through LLM-guided evolutionary operators (Wu et al., 2025). Let $\text{op} \in \{\text{mutation}, \text{crossover}\}$ denote the

operator type and Γ denote structural constraints (e.g., fixed layout structure, constrained resolution range). The offspring generation follows:

$$\rho_{\text{new}} \sim p_{\text{LLM}}(\cdot \mid \rho_{\text{best}}, \Phi(\rho_{\text{best}}), \text{op}, \Gamma) \tag{5}$$

where p_{LLM} represents the distribution over code-level variations induced by the LLM. For mutation, $\text{op} =$ mutation and the LLM modifies visual encoding strategies of ρ_{best}; for crossover, $\text{op} =$ crossover and the LLM receives two parent functions $\rho_1, \rho_2 \in \mathcal{P}_t$ to combine complementary design elements. We then update the population as $\mathcal{P}_{t+1} \leftarrow \mathcal{P}_t \cup \{\rho_{\text{new}}\}$ and select the top-K individuals by fitness: $\mathcal{P}_{t+1} \leftarrow \text{TopK}(\mathcal{P}_{t+1}, K)$.

Evaluator. For each candidate function $\rho_i \in \mathcal{P}_t$, we compute its fitness $\Phi(\rho_i) = -\mathcal{J}(\pi_{\theta_{\rho_i}}; \mathcal{D})$ by conducting supervised fine-tuning to obtain $\pi_{\theta_{\rho_i}}$ (the fine-tuning details are provided in Appendix C). The evaluation metric balances solution feasibility and optimality as defined in Eq. (6):

$$\mathcal{J}(\pi_\theta; \mathcal{D}) = (1 - \text{Feas}(\mathcal{B})) + \lambda \cdot \text{Gap}(\mathcal{B}) \tag{6}$$

where $\mathcal{B}$ denotes a held-out evaluation set of instances, $\text{Feas}(\mathcal{B})$ measures the feasibility rate and $\text{Gap}(\mathcal{B})$ quantifies the mean optimality gap relative to reference solutions, following the metrics defined in Section 3. When $\lambda = 1$, feasibility and optimality are equally weighted.

4.3 Policy Training via Supervised Learning and Reinforcement Learning

We adopt a two-stage training framework that progressively equips VLMs with the ability to solve CO problems from visual inputs. In Stage 1, we apply supervised fine-tuning (SFT) to initialize the policy model with high-quality solution trajectories. The model's optimization capability is then refined through reinforcement learning in Stage 2, which addresses the over-greedy behavior and constraint violations observed in purely supervised training.

Stage 1: Policy Initialization. In this stage, we initialize the model π_θ by training it on expert solution trajectories $\{(q, s_q)\}$ generated by domain-specific solvers. The goal here is to establish basic visual-to-solution mapping capabilities and enable the model to generate valid solution formats. Given a training dataset $\mathcal{D}$ of problem-solution pairs, each pair consists of a VAI $\varphi(q)$ containing the rendered image and textual constraints, paired with the corresponding expert solution s_q. Writing $\psi_q : \mathcal{S}_q \to \mathcal{L}$ for the deterministic serialization of a solution into its natural-language form (the inverse of the parser δ_q in Section 3), the model autoregressively generates the solution sequence $y = \psi_q(s_q) = (y_1, \ldots, y_n)$ by minimizing the following loss function:

$$\mathcal{J}_{\text{SFT}}(\theta) = -\mathbb{E}_{(q,s_q)\sim\mathcal{D}} \left[\sum_{i=1}^{n} \log \pi_\theta(y_i \mid \varphi(q), y_{<i}) \right], \tag{7}$$

where $\mathcal{D}$ is the training distribution of problem-solution pairs, y_i denotes the i-th token in the solution sequence, and $y_{<i}$ represents all preceding tokens. For computational efficiency, we freeze the vision encoder and only train lightweight LoRA (Hu et al., 2022) adapters on the language decoder's attention and MLP modules.

Stage 2: Reinforcement Learning for Visual Optimization. We further optimize the policy via group relative policy optimization (GRPO) (Shao et al., 2024) to address feasibility limitations and enable the model to recognize and satisfy problem-specific constraints encoded in visual representations. Specifically, given an input VAI $\varphi(q)$, the behavior policy $\pi_{\theta_{\text{old}}}$ samples a group of G candidate solutions $\{\hat{x}_{q,1}, \ldots, \hat{x}_{q,G}\}$ where $\hat{x}_{q,i} \sim \pi_{\theta_{\text{old}}}(\cdot|\varphi(q))$, each scored by a verifiable reward function $R^\Pi : \mathcal{S} \to \mathbb{R}$. Instead of relying on a learned critic to estimate value functions, GRPO computes relative advantages through within-group comparisons, providing more computationally efficient and interpretable training signals. The relative advantage of each candidate is $A_i = (R^\Pi(\hat{x}_{q,i}) - \bar{R})/\sigma_R$, where $\bar{R}$ and σ_R are the mean and standard deviation of rewards within the group. We update the policy π_θ by maximizing the objective in Eq. (8):

$$\mathcal{J}_{\text{RL}}(\theta) = \mathbb{E}_{\varphi(q)\sim\mathcal{D}, \{\hat{x}_{q,i}\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|\varphi(q))} \left[\frac{1}{G} \sum_{i=1}^{G} \min\left(\omega_i A_i, \text{clip}(\omega_i, 1-\epsilon, 1+\epsilon) A_i\right) - \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \right], \tag{8}$$

where $\omega_i = \pi_\theta(\hat{x}_{q,i}|\varphi(q))/\pi_{\theta_{\text{old}}}(\hat{x}_{q,i}|\varphi(q))$ is the importance sampling ratio, and β controls the Kullback-Leibler (KL) divergence penalty with respect to the reference policy π_{ref}. This objective encourages the model to increase the likelihood of high-reward solutions while preventing drastic policy updates.

To guide the policy toward feasible and high-quality solutions, we design verifiable rewards that explicitly balance constraint satisfaction and objective optimization (Eq. (9)):

$$R^{\Pi}(\hat{x}_q) = \mathbb{I}[\hat{x}_q \in \mathcal{S}_{\text{valid}}] \cdot \left(\lambda_0 + \sum_{j=1}^{m_q} \lambda_j c_{q,j}(\hat{x}_q) + \frac{1}{1 + M_{\text{gap}}(\hat{x}_q)} \right), \tag{9}$$

where $\mathbb{I}[\cdot]$ is the indicator function, $\mathcal{S}_{\text{valid}} \subseteq \mathcal{S}_q$ denotes the set of solutions with valid format, $c_{q,j}(\hat{x}_q) \in \{0, 1\}$ verifies whether the j-th constraint is satisfied, $M_{\text{gap}}(\hat{x}_q)$ measures the optimality gap relative to the training solver's solution, and λ_j are constraint-specific weights. The reward naturally decomposes into feasibility rewards (constraint satisfaction) and optimality rewards (objective quality), encouraging the model to generate both feasible and high-quality solutions. Through this trial-and-error exploration guided by verifiable feedback, the VLM gradually learns to generate feasible, high-quality solutions directly from rendered problem representations.

4.4 Inference-Time Multi-View Augmentation

Compared with textual representations, visual inputs can be easily transformed into multiple semantics-preserving views, providing natural diversity at inference time. We exploit this property through visual augmentation (VA), instantiated for each problem family as follows. For routing problems (TSP and CVRP), we adopt the eight symmetries of the square, $\mathcal{T}_{\text{route}} = D_4$, consisting of four rotations and their reflections (Kwon et al., 2020). For graph problems (MIS and MVC), we sample a node permutation $\sigma \in S_n$ and transform the adjacency matrix as $A' = P_\sigma A P_\sigma^\top$, followed by re-rendering the permuted graph (Maron et al., 2018). Here S_k denotes the symmetric group on k elements and P_σ the corresponding permutation matrix. For scheduling problems, we apply a job permutation $\sigma_J \in S_J$ to JSSP and PFSP; for JSSP, we additionally relabel machines using $\sigma_M \in S_M$ while consistently remapping all operation-machine assignments (Schmidl et al., 2024). Because each transformation is a bijective relabeling of the underlying instance—a graph automorphism, a coordinate symmetry, or an index permutation—it induces a bijection on candidate solutions that carries feasible solutions to feasible solutions and leaves every objective value unchanged; the visual rendering, however, differs.

Given an input image I_q, these transformations generate N equivalent visual views $\{I_q^{(1)}, \ldots, I_q^{(N)}\}$. The VLM independently predicts a candidate solution for each view, $\{\hat{x}_q^{(1)}, \ldots, \hat{x}_q^{(N)}\}$. Each solution is mapped back to the original representation through the inverse transformation $\mathcal{T}^{-1}$, and the best feasible candidate is selected according to Eq. (10):

$$\hat{x}_q^* = \arg\min_i f_q\Big(\mathcal{T}^{-1}(\hat{x}_q^{(i)})\Big), \quad \text{s.t. } \mathcal{T}^{-1}(\hat{x}_q^{(i)}) \in \mathcal{F}_q. \tag{10}$$

5 Experiments

We build the visual CO solver by fine-tuning Qwen2.5-VL-7B-Instruct with LoRA, using the Unsloth framework for efficiency. Training consists of supervised learning on 500K synthetic instances per problem, followed by reinforcement learning on up to 3K instances. For inference, we use greedy decoding (temperature $= 0$) for single solutions and top-p sampling ($p = 0.7$) for Best-of-N ($N = 8$). We evaluate optimality gaps against Concorde (Applegate et al., 2011) for TSP, LKH for CVRP, OR-Tools (Cuvelier et al., 2023) for scheduling, and exact solvers for graph problems. More details are in Appendix C. Our code and data are available at https://anonymous.4open.science/r/VLM-CO/.

5.1 Baselines

We evaluate our visual CO solver against four categories of baselines. The first category includes general-purpose VLMs (both closed-source and open-source), such as OpenAI GPT-4o and GPT-4 Turbo, Anthropic Claude-Haiku-4.5 and Claude-Sonnet-5, and Qwen2.5-VL-72B and Qwen3-VL-32B. The second consists of vision-enabled reasoning models leveraging extended test-time

Table 1: Main comparison across six CO problems. fea. = feasibility rate (↑); opt. = optimality gap (%, ↓, mean±std). 100 test instances per problem. **Gray**: best per column; underline: second-best. Ties are marked on the lowest-ranked method.

Method	TSP		CVRP		MIS		MVC		PFSP		JSSP		Avg. Time
	fea.	opt.	fea.	opt.	fea.	opt.	fea.	opt.	fea.	opt.	fea.	opt.	
General-purpose VLMs													
GPT-4o	65%	27.9%±13.3	41%	128.8%±77	10%	8.3%±6.5	41%	28.8%±17.9	61%	28.8%±10.2	11%	44.2%±22.5	3.8s
GPT-4 Turbo	43%	57.8%±20.7	27%	144.8%±61.4	3%	16.7%±22.5	87%	71.5%±49.1	80%	20.0%±8.9	3%	102.6%±66.1	11.2s
Claude-Haiku-4.5	70%	43.0%±14.8	27%	87.0%±34.3	3%	7.0%±10.6	87%	65.4%±40.9	60%	18.0%±10.3	10%	42.2%±34.8	5.0s
Claude-Sonnet-5	58%	38.5%±18.6	35%	42.6%±26.6	20%	28.9%±36.7	45%	15.1%±10.8	63%	17.7%±8.6	3%	103.0%±59.4	21.7s
Qwen2.5-VL-72B	43%	152%±88.3	23%	172.9%±64.9	13%	36.2%±3.4	63%	57.2%±40.6	80%	17.6%±9.1	6%	53.8%±31.8	2.0s
Qwen3-VL-32B	37%	123.8%±75.9	7%	127.5%±62.1	5%	26.7%±11.9	37%	41.8%±34.9	100%	16.4%±9.3	3%	29.8%±24.4	2.4s
Reasoning Models													
GPT-5.6 Sol	87%	112.3%±64.4	70%	84.0%±74.2	80%	7.7%±18.4	77%	17.0%±35.6	100%	9.7%±7.4	20%	59.7%±38.9	45.3s
GPT-6-Astra	63%	31.8%±23.3	70%	49.7%±45.5	97%	3.4%±9.0	77%	3.5%±8.8	97%	9.8%±7.9	30%	41.7%±33.5	42.5s
Claude-Opus-5	37%	43.8%±22.9	17%	124.6%±104.5	7%	16.7%±21.7	20%	60.6%±40.2	87%	19.9%±10.7	10%	53.9%±38.2	21.8s
Claude-Fable-5.1	40%	59.4%±27.0	30%	100.4%±55.2	10%	23.7%±22.8	37%	38.3%±41.7	90%	17.7%±9.3	13%	35.1%±29.4	16.4s
Gemini-2.5-Pro	57%	10.3%±6.8	43%	9.3%±4.0	67%	3.5%±6.2	30%	2.9%±4.0	90%	20.2%±8.7	40%	30.1%±18.8	68.6s
GLM-4.5V	53%	146.6%±107.5	27%	158.2%±88.9	10%	31.6%±25.2	30%	79.1%±44.2	100%	16.3%±9.4	1%	25.9%±34.3	65.4s
Prompt Strategies													
AlgoVP	20%	32.6%±15.4	70%	81.7%±100.1	10%	21.6%±14.3	50%	3.8%±3.2	60%	7.6%±3.4	20%	71.7%±49.1	117.9s
VP-SGE	85%	36.7%±26.5	72%	46.2%±49.3	98%	3.2%±3.5	80%	1.6%±1.8	100%	6.1%±5.6	40%	53.6%±31.6	78.2s
VP-PHP	90%	48.8%±20.9	65%	98.8%±86.2	92%	3.6%±7.6	94%	3.8%±7.4	95%	4.5%±7.4	87%	38.6%±49.2	79.2s
MVLM	80%	43.9%±21.3	56%	123.7%±76.7	100%	14.1%±10.8	73%	14.2%±10.8	98%	14.1%±12.3	89%	79.3%±56.5	93.0s
End-to-End Strategies													
Ours (SFT)	69%	8.32%±5.6	62%	14.30%±9.9	60%	0.82%±2.2	75%	1.89%±3.2	100%	4.53%±3.1	78%	18.43%±13.5	4.6s
Ours (SFT+RL)	90%	8.25%±5.9	88%	11.62%±8.6	87%	1.97%±4.1	91%	2.31%±3.6	100%	2.31%±2.3	96%	13.52%±11.4	4.6s
Ours (SFT+RL+BoN)	100%	5.97%±4.1	100%	9.07%±6.3	**100%**	2.08%±4.3	100%	1.98%±3.1	100%	1.85%±2.5	**100%**	10.93%±8.9	7.6s
Ours (SFT+RL+VA)	**100%**	**5.69%±3.8**	**100%**	**8.50%±6.0**	95%	**2.03%±4.1**	**100%**	**1.87%±3.0**	**100%**	**1.78%±2.1**	98%	**10.73%±8.6**	8.4s

computation, including OpenAI GPT-5.6 Sol and GPT-6-Astra, Anthropic Claude-Opus-5 and Claude-Fable-5.1, Google Gemini-2.5-Pro, and Zhipu GLM-4.5V. The third comprises prompt-based optimization strategies adapted for visual problem instances, including AlgoVP (Ye et al., 2024) for iterative visual prompt search, visual self-guiding exploration (VP-SGE) (Iklassov et al., 2024) for program generation, visual progressive-hint prompting (V-PHP) (Zheng et al., 2023) for progressive refinement, and MVLM (Huang et al., 2025) for direct multimodal solving. The fourth covers domain-specific methods including OR-Tools (Cuvelier et al., 2023) and ant colony optimization (ACO) (Dorigo et al., 2006) for routing problems, alongside problem-specific heuristics such as nearest neighbor (NN) and farthest insertion (FI) for TSP, sweep and parallel savings (PS) for CVRP, Greedy and Degree-based methods for MIS and MVC, Palmer's and Nawaz-Enscore-Ham (NEH) (Nawaz et al., 1983) for PFSP, and dispatching rules including shortest processing time (SPT), first-in-first-out (FIFO), and apparent tardiness cost (ATC) for JSSP. Detailed descriptions of all baseline methods are provided in Appendix D.

5.2 Main Results

Table 1 presents the main results across three baseline categories: general-purpose VLMs, vision-enabled reasoning models, and prompt-based optimization methods.

General-purpose VLMs fall short in combinatorial constraint satisfaction. Table 1 shows that general-purpose VLMs achieve limited feasibility across the evaluated CO problems, highlighting their difficulty in satisfying combinatorial constraints without task-specific training. Models such as GPT-4o and Claude-Sonnet-5 achieve feasibility rates below 50% on routing and scheduling tasks, with CVRP and JSSP particularly challenging where capacity and precedence constraints dominate. On the other hand, when solutions are valid, optimality gaps remain wide across all problems, typically ranging from 20% to over 100% deviation from optimal solutions.

Reasoning models improve feasibility but fail at combinatorial search. GPT-6-Astra and Gemini-2.5-Pro achieve 97% and 67% feasibility on MIS, respectively, indicating that additional reasoning helps satisfy combinatorial constraints. However, the gains are smaller on routing and scheduling tasks, with feasibility below 70% on CVRP and 40% on JSSP. Even when solutions are valid, optimality gaps remain wide at over 30%, showing that reasoning helps satisfy constraints but struggles with combinatorial search. The cost is steep, with inference taking 42-69 seconds per instance versus 2-22 seconds for general-purpose VLMs.

Prompt-based strategies approach feasibility but sacrifice speed and robustness. VP-PHP and VP-SGE achieve 87–100% feasibility on most tasks, indicating that repeated prompting can effectively guide models toward valid solutions. However, solution quality remains limited, with

Table 2: Comparison with domain-specific methods across problem scales. Gap = optimality gap (%, ↓); Gap@K = percentage of instances with gap $< K\%$ (↑). Small/Medium/Large: routing (10–30/40–60/70–100 nodes), graph (10–30/40–60/70–100 nodes), scheduling ($5 \times 10/10 \times 15/15 \times 20$).

Problem	Method	Small				Medium				Large			
		Gap	@1	@5	@10	Gap	@1	@5	@10	Gap	@1	@5	@10
TSP	OR-Tools	0.82%	76%	96%	99%	2.59%	28%	86%	99%	3.59%	12%	80%	99%
	NN	19.36%	3%	5%	19%	24.13%	0%	0%	1%	26.19%	0%	0%	2%
	FI	2.27%	51%	82%	96%	4.41%	8%	64%	95%	4.86%	2%	53%	99%
	ACO	1.98%	48%	88%	100%	17.98%	0%	1%	6%	36.69%	0%	0%	0%
	Ours(BoN)	1.55%	44%	100%	100%	4.02%	17%	63%	92%	7.26%	0%	28%	78%
	Ours(VA)	1.48%	44%	100%	100%	3.84%	17%	71%	92%	6.92%	0%	33%	82%
CVRP	OR-Tools	3.60%	45%	69%	93%	7.87%	3%	24%	72%	8.84%	0%	15%	71%
	Sweep	18.36%	8%	18%	32%	20.59%	0%	1%	8%	22.07%	0%	0%	3%
	PS	3.95%	24%	72%	93%	5.67%	2%	50%	89%	6.12%	0%	41%	89%
	ACO	2.52%	44%	81%	96%	17.07%	0%	2%	14%	29.49%	0%	0%	0%
	Ours(BoN)	4.73%	27%	64%	86%	8.30%	10%	33%	70%	12.40%	0%	5%	49%
	Ours(VA)	4.36%	27%	73%	86%	7.77%	10%	35%	78%	11.67%	0%	7%	52%
MIS	Degree	5.89%	57%	57%	67%	7.52%	32%	42%	64%	9.61%	20%	33%	59%
	Greedy	2.56%	84%	84%	86%	2.79%	67%	71%	90%	3.36%	53%	60%	83%
	Ours(BoN)	0.61%	97%	97%	97%	2.02%	78%	85%	88%	3.29%	84%	94%	100%
	Ours(VA)	0.61%	97%	97%	97%	1.97%	81%	86%	89%	3.28%	82%	96%	100%
MVC	Approx	49.79%	0%	0%	0%	39.21%	0%	0%	0%	35.35%	0%	0%	0%
	Greedy	2.80%	71%	72%	89%	2.62%	44%	79%	99%	2.21%	33%	89%	100%
	Degree	3.53%	63%	63%	86%	2.78%	44%	78%	98%	2.63%	26%	83%	99%
	Ours(BoN)	1.98%	72%	76%	100%	2.59%	54%	75%	100%	2.21%	47%	84%	100%
	Ours(VA)	1.98%	72%	76%	100%	2.59%	54%	75%	100%	2.11%	51%	86%	100%
PFSP	Palmer's	30.52%	0%	0%	1%	30.41%	0%	0%	0%	30.68%	0%	0%	0%
	NEH	1.33%	53%	97%	99%	2.78%	11%	90%	100%	3.56%	0%	88%	100%
	Ours(BoN)	0.46%	70%	90%	100%	1.45%	60%	98%	100%	3.30%	26%	76%	100%
	Ours(VA)	0.42%	70%	100%	100%	1.40%	60%	98%	100%	2.99%	24%	83%	100%
JSSP	SPT	19.58%	2%	4%	17%	25.32%	0%	0%	1%	27.35%	0%	0%	0%
	FIFO	24.38%	2%	4%	12%	32.97%	0%	0%	1%	39.00%	0%	0%	0%
	ATC	20.71%	0%	12%	15%	24.30%	0%	0%	1%	27.99%	0%	0%	0%
	Ours(BoN)	4.14%	21%	61%	77%	12.28%	0%	8%	44%	20.14%	0%	0%	5%
	Ours(VA)	3.97%	20%	66%	80%	12.03%	0%	8%	50%	19.83%	0%	0%	5%

VP-PHP showing optimality gaps of 38–99% across problems. Code-generation methods such as VP-SGE are sensitive to generated hyperparameters, which are difficult for VLMs to tune. Moreover, iterative refinement substantially increases inference time to 78–118 seconds per instance. Overall, these methods improve feasibility, but their solution quality, robustness, and efficiency remain limited.

Our method achieves high feasibility and optimality through task-specific training. Our task-specific training consistently improves both feasibility and solution quality. The SFT-only variant already surpasses reasoning and prompt-based baselines in optimality while matching or exceeding their feasibility on most tasks. Adding reinforcement learning raises feasibility to 98–100% on five of six problems and reduces optimality gaps by 20–40% over SFT alone. Best-of-N sampling with $N = 8$ further improves feasibility while preserving strong optimality. Our model achieves 95–100% feasibility and the smallest optimality gaps among all baselines. Despite using only a 7B model, inference remains under 10 seconds, making it 5–12$\times$ faster than prompt-based strategies and 4–7$\times$ faster than reasoning models.

5.3 Comparison with Domain-specific Methods

Most classical CO solvers are tailored to specific problem classes using heuristic or metaheuristic strategies. We compare our method against these domain-specific baselines on 100 instances per setting across three problem scales, reporting both the mean optimality gap and the distribution of solution quality through Gap@K. Table 2 details the instance sizes and metric definitions.

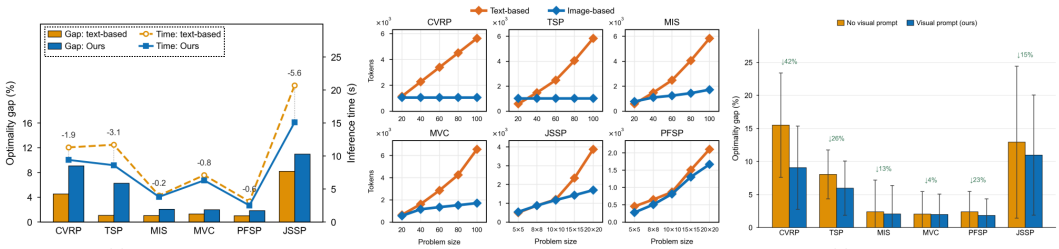
As shown in the table, our model can outperform simple domain-specific algorithms, especially on smaller instances. On graph problems (MIS, MVC), our method achieves strong Gap@1 rates, indicating near-optimal solutions on a significant portion of instances. On scheduling problems (PFSP), our model shows competitive performance across all scales. However, on routing problems (TSP, CVRP) and JSSP, the performance gap widens on larger instances, revealing challenges in scaling visual reasoning to complex problem structures.

5.4 Comparison with Text-based Solvers

We compare the visual solver with text-based LLM solvers (Jiang et al., 2026) that receive structured textual representations instead of rendered images. For a fair comparison, we use the SFT+RL+BoN variant. As shown in Table 3, both approaches achieve 100% feasibility across all six problems. While the text-based solver generally yields lower optimality gaps, the difference varies by problem type:

Table 3: Comparison between text-based and visual CO solvers under identical configuration (SFT+RL+BoN). Each cell shows feasibility rate (↑) and optimality gap (%, ↓, mean±std). Δ rows report the difference (visual minus text-based).

Method	TSP	CVRP	MIS	MVC	PFSP	JSSP
Text-based	100% 1.07%±0.9	100% 4.53%±3.5	100% 1.04%±3.4	100% 1.29%±2.2	100% 1.03%±1.1	100% 8.20%±6.3
Ours (SFT+RL+BoN)	100% 5.97%±4.1	100% 9.07%±6.3	100% 2.08%±4.3	100% 1.98%±3.1	100% 1.85%±2.5	100% 10.93%±8.9
Δ Gap (%, ↓)	(+4.90)	(+4.54)	(+1.04)	(+0.69)	(+0.82)	(+2.78)
Δ Fea. (%, ↑)	(0)	(0)	(0)	(0)	(0)	(0)



(a) Running time (b) Token consumption (c) Ablation study

Figure 3: Comparison with text-based solvers and visual prompt design ablation.

it is small on graph problems (MVC: +0.69%; MIS: +1.04%) but larger on routing problems (TSP: +4.90%; CVRP: +4.54%), where precise geometric information is more important. This likely stems from textual inputs providing exact node coordinates, whereas visual inputs encode only relative spatial positions without explicit coordinates. Despite this gap, Figure 3(a) and (b) shows that the visual solver requires substantially less inference time and fewer input tokens, offering a favorable trade-off between solution quality and efficiency.

5.5 Ablation on Visual Prompt Design

The self-evolutionary visual prompt optimization introduces informative cues such as spatial coloring and degree-based shading into rendered problem instances. As shown in Figure 3(c), removing these evolved visual prompts consistently reduces feasibility by 7–15% and increases optimality gaps by 1–7% across all tested problems (Appendix E.4). These results show that evolved visual prompts provide effective structural guidance for high-quality solution generation.

5.6 Generalization to Out-of-Distribution Instances

To evaluate generalization, we test three out-of-distribution (OOD) node distributions (uniform, Gaussian mixture, clustered) for routing and real-world benchmarks including TSPLIB (Zhu et al., 2026) for TSP and Taillard's suite (Graviers et al., 2025) for JSSP. Results show slight performance drops on OOD distributions, demonstrating robust cross-distribution generalizability. On real-world benchmarks, our method achieves competitive performance with traditional heuristics and text-based solvers, indicating generalization to real-world scenarios. Details are in Appendices E.7 and E.8.

6 Conclusion

In this paper, we propose a vision-language model framework as an end-to-end combinatorial optimization solver. We consider three major problem families: routing (TSP, CVRP), graph optimization (MIS, MVC), and scheduling (JSSP, PFSP). Through a two-stage training strategy combining supervised fine-tuning and reinforcement learning with verifiable rewards, our 7B VLM achieves 100% feasibility and optimality gaps of 1.78 to 10.93% across all six problems. Compared to text-based solvers, our visual approach demonstrates competitive performance while offering substantially reduced inference time and token consumption. Our method also outperforms commonly used heuristics on most problem instances. Furthermore, we demonstrate that VLMs possess reasoning capabilities competitive with LLMs for combinatorial optimization tasks. In future work, we plan to explore automated visual structure design, improve generalization across problem types and scales, and extend the framework to larger and more complex graph instances.

AI USE STATEMENT

Generative AI tools were used solely to assist with English language polishing, grammatical correction, and stylistic refinement of author-written text. They were not used to formulate research ideas, design the methodology, conduct experiments, analyze data, generate scientific claims, or draw conclusions. All AI-assisted revisions were carefully reviewed, edited, and verified by the authors, who take full responsibility for the accuracy, originality, and integrity of the final manuscript.

REFERENCES

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pp. 2425–2433, 2015.

David L Applegate, Robert E Bixby, Vašek Chvátal, and William J Cook. The traveling salesman problem: a computational study. In *The traveling salesman problem*. Princeton university press, 2011.

Thibaut Cuvelier, Frederic Didier, Vincent Furnon, Steven Gay, Sarah Mohajeri, and Laurent Perron. Or-tools' vehicle routing solver: A generic constraint-programming solver with heuristic search for routing problems. In *24e congrès annuel de la société française de recherche opérationnelle et d'aide à la décision*, 2023.

Francesca Da Ros, Michael Soprano, Luca Di Gaspero, and Kevin Roitero. Large language models for combinatorial optimization: A systematic review. *ACM Computing Surveys*, 58(11):1–53, 2026.

Marco Dorigo, Mauro Birattari, and Thomas Stützle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006.

Mohammed Elhenawy, Ahmad Abutahoun, Taqwa I. Alhadidi, Ahmed Jaber, Huthaifa I. Ashqar, Shadi Jaradat, Ahmed Abdelhay, Sebastien Glaser, and Andry Rakotonirainy. Visual reasoning and multi-agent approach in multimodal large language models (MLLMs): Solving TSP and mTSP combinatorial challenges. *Machine Learning and Knowledge Extraction*, 6(3):1894–1920, 2024.

Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3 (4):128–135, 1999.

Marc-Emmanuel Coupvent des Graviers, Lotfi Kobrosly, Christophe Guettier, and Tristan Cazenave. Updating lower and upper bounds for the job-shop scheduling problem test instances. *arXiv preprint arXiv:2504.16106*, 2025.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.

Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12(966-980):1, 2017.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Yuxiao Huang, Wenjie Zhang, Liang Feng, Xingyu Wu, and Kay Chen Tan. How multimodal integration boost the performance of llm for optimization: Case study on capacitated vehicle routing problems. In *2025 IEEE Symposium for Multidisciplinary Computational Intelligence Incubators (MCII)*, pp. 1–7. IEEE, 2025.

Zangir Iklassov, Yali Du, Farkhad Akimov, and Martin Takáč. Self-guiding exploration for combinatorial problems. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 130569–130601. Curran Associates, Inc., 2024. doi: 10.52202/079017-4150. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/eb9120be0dcaac0aec66d2e75deb69dd-Paper-Conference.pdf.

Xia Jiang, Yaoxin Wu, Chenhao Zhang, and Yingqian Zhang. Droc: Elevating large language models for complex vehicle routing via decomposed retrieval of constraints. In *International Conference on Learning Representations*, volume 2025, pp. 46731–46752, 2025.

Xia Jiang, Yaoxin Wu, Minshuo Li, Zhiguang Cao, and Yingqian Zhang. Large language models as end-to-end combinatorial optimization solvers. *Advances in Neural Information Processing Systems*, 38:164787–164826, 2026.

Pawel Jan Kalczynski and Jerzy Kamburowski. On the neh heuristic for minimizing the makespan in permutation flow shops. *Omega*, 35(1):53–60, 2007.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution, 2022. *URL https://arxiv.org/abs/2202.10054*, 28.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in neural information processing systems*, 33:21188–21198, 2020.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 3045–3059, 2021.

Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhao-Xiang Zhang. Sheetcopilot: Bringing software productivity to the next level through large language models. *Advances in Neural Information Processing Systems*, 36:4952–4984, 2023a.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pp. 19730–19742. PmLR, 2023b.

Xue Li, Yiyou Sun, Wei Cheng, Yinglun Zhu, and Haifeng Chen. Chain-of-region: Visual language models need details for diagram analysis. In *International Conference on Learning Representations*, volume 2025, pp. 28547–28563, 2025.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *arXiv preprint arXiv:2401.02051*, 2024a.

Fuxiao Liu, Xiaoyang Wang, Wenlin Yao, Jianshu Chen, Kaiqiang Song, Sangwoo Cho, Yaser Yacoob, and Dong Yu. Mmc: Advancing multimodal chart understanding with large-scale instruction tuning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1287–1310, 2024b.

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 *International Conference on Learning Representations*, volume 2024, pp. 23439–23554, 2024.

Yueen Ma, Zixing Song, Yuzheng Zhuang, Jianye Hao, and Irwin King. A survey on vision–language–action models for embodied ai. *IEEE Transactions on Neural Networks and Learning Systems*, 2026.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 2263–2279, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.177. URL https://aclanthology.org/2022.findings-acl.177/.

M Jehanzeb Mirza, Mengjie Zhao, Zhuoyuan Mao, Sivan Doveh, Wei Lin, Paul Gavrikov, Michael Dorkenwald, Shiqi Yang, Saurav Jha, Hiromi Wakaki, et al. Glov: Guided large language models as implicit optimizers for vision language models. *arXiv preprint arXiv:2410.06154*, 2024.

Vijaykumar Yogesh Muley. Mathematical programming for modeling expression of a gene using gurobi optimizer to identify its transcriptional regulators. In *Modeling transcriptional regulation: Methods and protocols*, pp. 99–113. Springer, 2021.

Muhammad Nawaz, E Emory Enscore Jr, and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

Fernando Peres and Mauro Castelli. Combinatorial optimization problems and metaheuristics: Review, challenges, design, and development. *Applied Sciences*, 11(14):6449, 2021.

Bernardino Romera-Paredes et al. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

Daniel J. Rosenkrantz, Richard E. Stearns, and Philip M. Lewis. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977.

Christoph Schmidl, Thiago D. Simão, and Nils Jansen. A supervised learning approach to robust reinforcement learning for job shop scheduling. In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence*, pp. 1324–1335, 2024. doi: 10.5220/0012473600003636.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Yicheng Tao, Yuante Li, Yao Qin, and Yepang Liu. Retrieval-augmented code generation: A survey with focus on repository-level approaches. *arXiv preprint arXiv:2510.04905*, 2025.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

Chenguang Wang, Zhouliang Yu, Stephen McAleer, Tianshu Yu, and Yaodong Yang. ASP: Learn a universal neural solver! *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(6): 4102–4114, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Yanbin Wei, Shuai Fu, Weisen Jiang, Zejian Zhang, Zhixiong Zeng, Qi Wu, James Kwok, and Yu Zhang. GITA: Graph to visual and textual integration for vision-language graph reasoning. *Advances in Neural Information Processing Systems*, 37:44–72, 2024.

Xuan Wu, Di Wang, Chunguo Wu, Lijie Wen, Chunyan Miao, Yubin Xiao, and You Zhou. Efficient heuristics generation for solving combinatorial optimization problems using large language models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 3228–3239, 2025.

Yi Xu, Chengzu Li, Han Zhou, Xingchen Wan, Caiqi Zhang, Anna Korhonen, and Ivan Vulić. Visual planning: Let's think only with images. In *International Conference on Learning Representations*, volume 2026, pp. 36706–36739, 2026.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pp. 12028–12068, 2024.

Xin-She Yang. An introduction with metaheuristic applications. *Hoboken, New Jersey, USA: John Wiley & Sons*, 2010.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. *Advances in neural information processing systems*, 37:43571–43608, 2024.

Zhuoli Yin, Yi Ding, Reem Khir, and Hua Cai. Vitsp: A vision language models guided framework for solving large-scale traveling salesman problems. In *International Conference on Learning Representations*, volume 2026, pp. 132572–132599, 2026.

Lingfeng Zhang, Xiaoshuai Hao, Qinwen Xu, Qiang Zhang, Xinyao Zhang, Pengwei Wang, Jing Zhang, Zhongyuan Wang, Shanghang Zhang, and Renjing Xu. Mapnav: A novel memory representation via annotated semantic maps for vlm-based vision-and-language navigation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13032–13056, 2025.

Wentao Zhang, Jingyuan Wang, Zetong Zhou, Jiahao Ji, and Junjie Wu. See to solve: A geometry-aware vision-language agent for real-world routing problem. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 12904–12915, 2026.

Jie Zhao and Kang Hao Cheong. Visual evolutionary optimization on graph-structured combinatorial problems with MLLMs: A case study of influence maximization. *IEEE Transactions on Evolutionary Computation*, 2025.

Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. Progressive-hint prompting improves reasoning in large language models. *arXiv preprint arXiv:2304.09797*, 2023.

Jianghan Zhu, Yaoxin Wu, Zhuoyi Lin, Zhengyuan Zhang, Haiyan Yin, Zhiguang Cao, Senthilnath Jayavelu, and Xiaoli Li. Bridging synthetic and real routing problems via llm-guided instance generation and progressive adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 36591–36599, 2026.

Chengke Zou, Xingang Guo, Rui Yang, Junyu Zhang, Bin Hu, and Huan Zhang. Dynamath: A dynamic visual benchmark for evaluating mathematical reasoning robustness of vision language models. In *International Conference on Learning Representations*, volume 2025, pp. 48337–48383, 2025.

A Problem Class Definitions

We investigate six visual combinatorial optimization (CO) problem classes spanning routing, scheduling, and graph optimization. For each problem class, we provide its mathematical formulation, visual instance generation procedure, reference solution construction for supervised fine-tuning (SFT), and reward design for reinforcement learning (RL).

A.1 V-TSP

A.1.1 Problem Definition

The V-TSP extends the conventional TSP formulation by representing problem instances through visual observations rather than explicit numerical descriptions, such as coordinate lists or distance matrices. Given a rendered image containing n cities and their spatial configurations, the objective is to find a tour $\tau = (v_0, v_1, \ldots, v_{n-1}, v_0)$ where each city $v_i \in V = \{v_0, v_1, \ldots, v_{n-1}\}$ is visited exactly once, the tour forms a closed loop starting and ending at v_0, while minimizing the total travel distance:

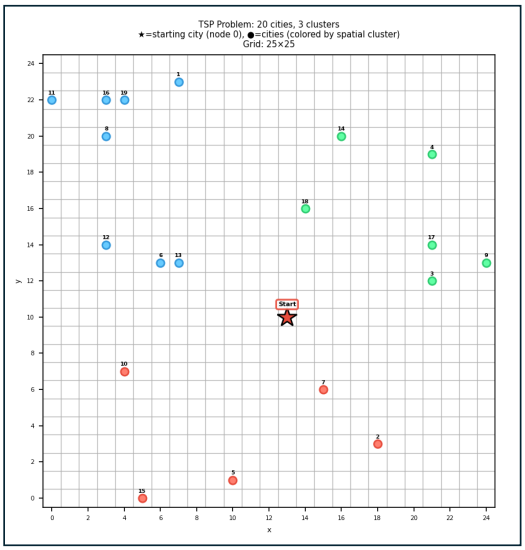


Figure 4: V-TSP.

$$\min_{\tau} \sum_{i=0}^{n-1} d(v_{\tau(i)}, v_{\tau((i+1) \bmod n)}) \tag{11}$$

where $d(v_i, v_j)$ denotes the Euclidean distance between two cities. Unlike conventional TSP solvers that directly access explicit numerical coordinates, V-TSP requires the model to infer spatial positions and pairwise relationships from the visual input. The model receives only the rendered image and generates the corresponding tour sequence as textual output.

A.1.2 VAI Generation

Visual-attributed instances (VAIs) are constructed by transforming numerical problem data into visually rendered representations. For V-TSP, diverse problem instances are first generated by randomly sampling the number of cities $n \sim \mathcal{U}[10, 100] \cap \mathbb{Z}$ and their coordinates $(x_i, y_i) \sim \mathcal{D}$ where $\mathcal{D}$ is one of three distributions: uniform distribution $\mathcal{U}[1, 1000]^2$, or two Gaussian mixture (GM) distributions parameterized by $(c = 2, l = 5)$ and $(c = 3, l = 10)$. We then render each numerical instance $\mathcal{I} = \{(x_i, y_i)\}_{i=0}^{n-1}$ into a 2-D scatter plot image $\mathcal{V}(\mathcal{I})$ showing all cities with their labels, enabling the model to perceive spatial relationships visually. Reference solutions for supervised fine-tuning are obtained using the LKH-3 solver, which produces optimal solutions for instances with 10-100 cities.

An example VAI used for SFT is shown below:

Example VAI of V-TSP

Input: The image is presented in Figure 4.

Instruction: This is a TSP instance with 20 cities, where the starting city (node 0) is marked with a star symbol ⋆ and the other cities (nodes 1–19) are represented as circles colored by spatial cluster with their node IDs labeled above. The goal is to find a tour starting from node 0, visiting all 20 cities exactly once, and returning to node 0 while minimizing the total travel distance. Output format — no other text: Route: [0, n1, n2, ..., 0], Objective: ⟨total_distance⟩

Output: Route: [0, 3, 9, 17, 4, 14, 18, 1, 19, 16, 11, 8, 12, 13, 6, 10, 15, 5, 2, 7, 0] Objective: 3970.801

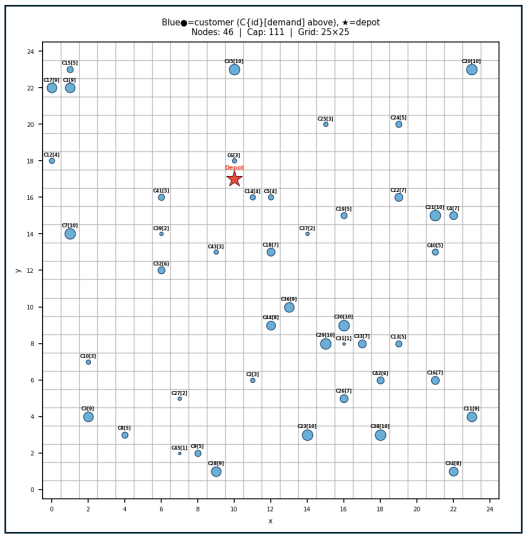


Figure 5: V-CVRP.

A.1.3 Reward Function

The reward function for V-TSP is designed to evaluate both constraint satisfaction and solution quality. The feasibility reward R_f quantifies the extent to which a proposed tour satisfies the problem constraints:

$$R_f = \alpha_0 + \alpha_1 c_1 + \alpha_2 c_2 \tag{12}$$

where the constraint indicators are defined as: $c_1 = 1$ if all cities are visited exactly once (0 otherwise), and $c_2 = 1$ if the tour forms a closed loop returning to the starting city (0 otherwise). The reward coefficients are set to $\alpha_0 = 0.2$, $\alpha_1 = 0.5$, and $\alpha_2 = 0.3$, prioritizing the node coverage constraint.

A.2 V-CVRP

A.2.1 Problem Definition

Given a rendered image containing a depot (node 0) and n customers with their spatial configurations and demand values, the objective is to find a set of routes $\{\tau_1, \tau_2, \ldots, \tau_K\}$ where each route τ_k starts and ends at the depot, each customer is visited exactly once across all routes ($\bigcup_{k=1}^{K} \tau_k = \{1, 2, \ldots, n\}$ with $\tau_k \cap \tau_{k'} = \emptyset$ for $k \neq k'$), and the total demand on each route does not exceed the vehicle capacity Q (i.e., $\sum_{i \in \tau_k} q_i \leq Q$ for all k), such that the total travel distance is minimized:

$$\min \sum_{k=1}^{K} \sum_{i=0}^{|\tau_k|-1} d(v_{\tau_k(i)}, v_{\tau_k(i+1)}) \tag{13}$$

where $d(v_i, v_j)$ denotes the Euclidean distance between nodes and q_i is the demand of customer i. Unlike conventional CVRP solvers that directly access coordinates and demand values, the model must infer spatial positions, demands, and routing decisions from the visual input.

An example VAI used for SFT is shown below:

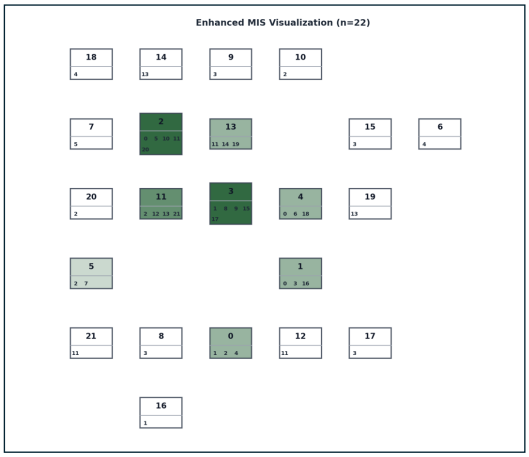


Figure 6: V-MIS.

Example VAI of V-CVRP

Input: The image is presented in Figure 5.

Instruction: This is a CVRP instance with 1 depot and 41 customer nodes. Each blue circle represents a customer (labeled with node ID and demand in brackets), and the red star marks the depot (node 0). The circle size is proportional to customer demand. The vehicle capacity is 111 units. Each vehicle must start and end at the depot, visit customers assigned to it exactly once, and ensure the total demand on its route does not exceed the capacity. The goal is to assign all customers to vehicle routes such that the total travel distance is minimized. Output format — no other text: Routes: [[0, ..., 0], [0, ..., 0], ...], Objective: ⟨total_distance⟩

Output: Routes: [[0, 6, 35, 1, 15, 17, 12, 7, 10, 3, 8, 27, 45, 9, 28, 23, 2, 32, 39, 41, 0], [0, 25, 24, 20, 4, 40, 21, 22, 19, 37, 5, 0], [0, 43, 44, 29, 26, 42, 38, 34, 11, 16, 13, 33, 31, 30, 36, 18, 14, 0]], Objective: 6643.76

A.2.2 VAI Generation

We follow the same spatial distribution scheme as V-TSP, randomly sampling $n \sim \mathcal{U}[10, 100] \cap \mathbb{Z}$ customers and coordinates $(x_i, y_i) \sim \mathcal{D}$ from uniform or Gaussian mixture distributions. Each instance is rendered as an image depicting the depot (node 0, highlighted with a distinct marker) and customer nodes with their demand values q_i annotated. Customer demands are randomly sampled from $[1, 10] \cap \mathbb{Z}$, and the vehicle capacity Q is set to the average demand across all customers. Near-optimal reference solutions are obtained using the LKH-3 solver.

A.2.3 Reward Function

The feasibility reward of V-CVRP is $R_f = \alpha_0 + \alpha_1 c_1 + \alpha_2 c_2 + \alpha_3 c_3 + \alpha_4 c_4$, where $\alpha_0 = 0.1$, $\alpha_1 = 0.1$, $\alpha_2 = 0.35$, $\alpha_3 = 0.35$, and $\alpha_4 = 0.1$. Specifically, $c_1 = 1$ if all routes start and end at the depot (otherwise 0); $c_2 = 1$ if the capacity constraint is satisfied for all routes (otherwise 0); $c_3 = 1$ if all customer nodes are visited (otherwise 0); $c_4 = 1$ if no customer is visited more than once (otherwise 0).

A.3 V-MIS

A.3.1 Problem Definition

Given a rendered image of an undirected graph $G = (V, E)$ where $V = \{0, 1, \ldots, n-1\}$ is the set of vertices and E is the set of edges, the objective is to find the largest subset $S \subseteq V$ where no

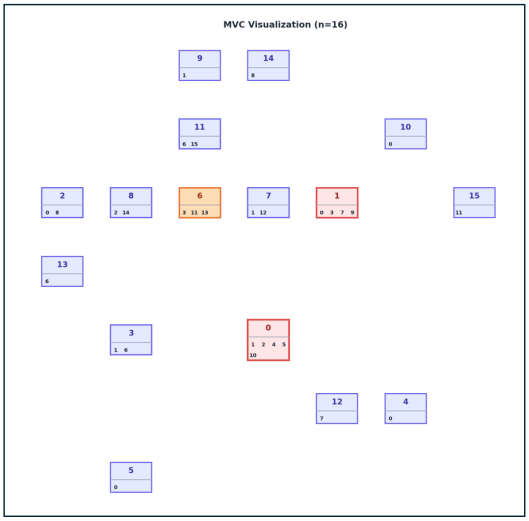


Figure 7: V-MVC.

two vertices in S are adjacent (i.e., for any $u, v \in S$, $(u, v) \notin E$), such that the cardinality of the independent set is maximized:

$$\max |S| \tag{14}$$

Unlike conventional MIS solvers that directly access the adjacency matrix or edge list, V-MIS requires the model to infer graph structure and vertex relationships from the visual input. The model receives only the rendered graph image and generates the corresponding independent set as textual output.

A.3.2 VAI Generation

For the instances of V-MIS, we generate graphs using the Python networkx package and render them as node-link diagrams. We randomly choose the graph type from two options: Erdős-Rényi graph (ER) or Barabási-Albert (BA) graph. If ER graph is chosen, the edge probability is sampled from [0.1, 0.4]; if BA graph is chosen, the number of edges each new node attaches to is randomly sampled from $[1, 4] \cap \mathbb{Z}$. The number of nodes is sampled from $[10, 100] \cap \mathbb{Z}$. Each graph is rendered as an image with nodes and edges clearly visible. The optimal solution is obtained by Gurobi.

An example VAI used for SFT is shown below:

Example VAI of V-MIS

Input: The image is presented in Figure 6.

Instruction: This is a Maximum Independent Set (MIS) instance with 22 nodes. Each node is displayed as a box with its ID in the top section and the IDs of its adjacent (neighbor) nodes listed in the bottom section (up to 4 neighbor IDs per row). Each node is colored based on its degree, where darker colors indicate higher degree (more connections) and lighter colors indicate lower degree (fewer connections). The goal is to find the largest subset of vertices such that no two vertices in the subset are connected by an edge. Output format — no other text: Set: [list of vertex IDs], Objective: ⟨set_size⟩

Output: Set: [0, 6, 7, 8, 9, 10, 12, 14, 15, 16, 17, 18, 19, 20, 21] Objective: 15

A.3.3 Reward Function

The feasibility reward is $R_f = \alpha_0 + \alpha_1 c_1$, where $\alpha_0 = 0.2$ and $\alpha_1 = 0.8$. The value of c_1 is 1 if no two vertices in the solution are adjacent (otherwise 0).

A.4 V-MVC

A.4.1 Problem Definition

The V-MVC problem seeks to identify a minimum-cardinality vertex cover in a graph represented visually. Given a rendered image of an undirected graph $G = (V, E)$, the task is to select a subset $C \subseteq V$ with the smallest possible size such that each edge $(u, v) \in E$ is incident to at least one vertex in C. The optimization objective is:

$$\min |C| \quad \text{subject to} \quad \forall (u, v) \in E : u \in C \text{ or } v \in C \tag{15}$$

The model must extract the graph topology from the image and output a valid vertex cover, rather than working with explicit edge lists or adjacency representations.

A.4.2 VAI Generation

We follow the same graph generation and visualization process as V-MIS. The optimal solution is also produced by Gurobi.

An example VAI used for SFT is shown below:

Example VAI of V-MVC

Input: The image is presented in Figure 7.

Instruction: This is a Minimum Vertex Cover (MVC) instance with 16 nodes. Each node is displayed as a box with its ID in the top section and the IDs of its adjacent (neighbor) nodes listed in the bottom section (up to 4 neighbor IDs per row). Each node is colored based on its degree, where darker colors indicate higher degree (more connections) and lighter colors indicate lower degree (fewer connections). The goal is to find the smallest subset of vertices such that every edge in the graph has at least one endpoint in the subset. Output format — no other text: Set: [list of vertex IDs], Objective: ⟨set_size⟩

Output: Output: Set: [0, 1, 6, 7, 8, 11], Objective: 6

A.4.3 Reward Function

The feasibility reward of V-MVC is $R_f = \alpha_0 + \alpha_1 c_1$, where $\alpha_0 = 0.2$ and $\alpha_1 = 0.8$. The value of c_1 is 1 if all edges are covered by the solution (otherwise 0).

A.5 V-JSSP

A.5.1 Problem Definition

In the V-JSSP, we are given a rendered image depicting J jobs and M machines, where each job j consists of a sequence of operations $O_j = (o_{j,1}, o_{j,2}, \ldots, o_{j,m_j})$ that must be executed in order. Each operation $o_{j,k}$ requires processing on a designated machine $\mu(j, k)$ for a duration $p(j, k)$. The scheduling objective is to assign start times to all operations to minimize the makespan $C_{\max}$:

$$\min C_{\max} = \max_{j \in \{0, \ldots, J-1\}} C_j \tag{16}$$

where C_j denotes the completion time of job j. The schedule must respect precedence constraints (operations within each job execute sequentially) and machine capacity constraints (each machine processes at most one operation at any time). The model interprets the visual representation of job-operation-machine assignments and produces a feasible schedule.

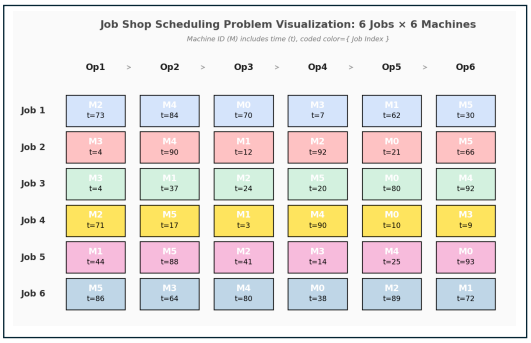


Figure 8: V-JSSP.

A.5.2 VAI Generation

Each JSSP instance is rendered as a Gantt chart skeleton showing the machine layout and job structure. We randomly sample the number of jobs J and machines M from [5, 20], and sample processing times from $[1, 100] \cap \mathbb{Z}$. Each job is randomly assigned to machines with specific operation sequences. The near-optimal solution is calculated based on the CP-SAT solver of OR-Tools with a time limit of 300s.

An example VAI used for SFT is shown below:

Example VAI of V-JSSP

Input: The image is presented in Figure 8.

Instruction: This is a Job Shop Scheduling Problem (JSSP) instance with 8 jobs and 5 machines. Each row represents a job (J0, J1, ..., J7) and each column represents an operation step. Each cell displays the machine ID (e.g., M2) and processing time (e.g., t=56) for that operation, with different machines color-coded. Arrows indicate that operations within a job must be processed strictly from left to right. Each machine can process only one operation at a time. The goal is to find a schedule that minimizes the makespan (maximum completion time). Output format — no other text: Schedule: [[job order on M0], [job order on M1], ...], Makespan: ⟨makespan_value⟩

Output: Schedule: [[10, 6, 4, 7, 5, 8, 1, 3, 2, 9, 0, 11], ..., [0, 3, 4, 5, 8, 2, 9, 6, 11, 1, 10, 7]], Makespan: 13106

A.5.3 Reward Function

The feasibility reward of V-JSSP is $R_f = \alpha_0 + \alpha_1 c_1 + \alpha_2 c_2 + \alpha_3 c_3$, where $\alpha_0 = 0.2$, $\alpha_1 = 0.2$, $\alpha_2 = 0.2$, and $\alpha_3 = 0.4$. The value of c_1 is 1 if all jobs are scheduled (otherwise 0), $c_2 = 1$ if there are no machine conflicts (otherwise 0), and $c_3 = 1$ if the precedence constraint is satisfied (otherwise 0).

A.6 V-PFSP

A.6.1 Problem Definition

The V-PFSP considers a flow shop environment with J jobs and M machines arranged in series, where all jobs follow the same routing through the machines. Given a rendered image displaying the processing time matrix $P = [p_{j,m}]$ (where $p_{j,m}$ is the time required for job j on machine m), the objective is to determine a permutation ς of jobs that minimizes the total completion time. Formally, let $C_{j,m}(\varsigma)$ denote the completion time of job $\varsigma(j)$ on machine m. The makespan under permutation ς is:

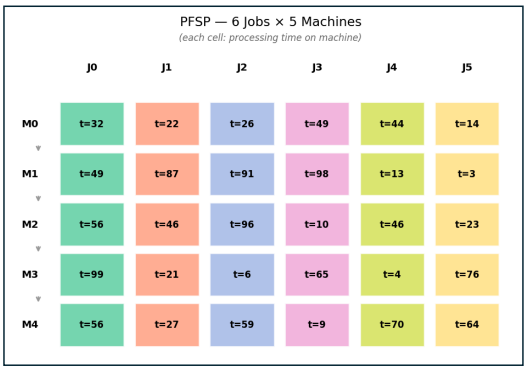


Figure 9: V-PFSP.

$$C_{\max}(\varsigma) = \max_j C_{j,M}(\varsigma) \tag{17}$$

The goal is to find $\varsigma^* = \arg\min_\varsigma C_{\max}(\varsigma)$. The constraint is that all jobs must be processed in the same order on every machine. The model must visually parse the processing time matrix and output the optimal job sequence.

A.6.2 VAI Generation

The instance of V-PFSP is rendered as an image depicting the flow shop structure with machines and jobs. The processing time matrix has J rows and M columns, with each element representing the processing time for a job on a machine. We randomly sample J and M from [5, 20], and sample processing times from $[1, 100] \cap \mathbb{Z}$. The near-optimal solutions are obtained by the Q-learning-based iterated greedy (QIG) method.

An example VAI used for SFT is shown below:

Example VAI of V-PFSP

Input: The image is presented in Figure 9.

Instruction: This is a Permutation Flow Shop Problem (PFSP) instance with 6 jobs and 5 machines. The image shows a processing time matrix where each row represents a machine (M0, M1, ..., M4) and each column represents a job (J0, J1, ..., J5). Each cell is displayed as a colored box containing the processing time (t=value) for that job on that machine, where different jobs are represented by distinct colors for visual distinction. All jobs must be processed on each machine in the same order. Each machine can process only one job at a time, and each job can be processed by only one machine at a time. The goal is to find the job processing order that minimizes the makespan (maximum completion time). Output format — no other text: Order: [job sequence], Objective: ⟨makespan_value⟩

Output: Order: [5, 4, 3, 6, 2, 1], Objective: 524

A.6.3 Reward Function

The feasibility reward of V-PFSP is $R_f = \alpha_0 + \alpha_1 c_1$, where $\alpha_0 = 0.2$ and $\alpha_1 = 0.8$, and $c_1 = 1$ if the solution sequence contains each job exactly once (otherwise 0).

B Low-Rank Adaptation

We adopt Low-Rank Adaptation (LoRA) (Hu et al., 2022) to adapt vision-language models to CO tasks, motivated by three challenges of full fine-tuning at billion-parameter scale: (1) *Memory*

overhead – optimizer states (momentum, variance) scale linearly with model size; (2) *Storage costs* – a separate checkpoint per task approaches the size of the base model; and (3) *Catastrophic forgetting* (French, 1999) – unrestricted updates risk degrading the visual understanding and reasoning acquired during pre-training.

Among parameter-efficient fine-tuning approaches (He et al., 2021; Lester et al., 2021), LoRA matches full fine-tuning while updating fewer than 1% of parameters (Hu et al., 2022). We apply it exclusively to the language decoder, keeping the vision encoder frozen: pre-trained encoders (Li et al., 2023b) already perceive CO instances well, and fine-tuning them risks disrupting these representations and harming generalization (Kumar et al.). Adapting the decoder alone suffices to learn CO-specific reasoning over frozen visual features, preserving visual understanding, accelerating convergence, and focusing adaptation on the mapping from visual features to symbolic solutions.

LoRA operates on the principle that task-specific adaptations reside in a low-dimensional subspace of the full parameter space. Consider a pre-trained weight matrix $W_0 \in \mathbb{R}^{d\times k}$ from any layer in the VLM (e.g., attention projections or feed-forward transformations). Rather than learning an unrestricted update $\Delta W \in \mathbb{R}^{d\times k}$, LoRA constrains the adaptation to a low-rank factorization:

$$W = W_0 + \Delta W = W_0 + BA \tag{18}$$

where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$ with rank $r \ll \min(d, k)$. The key insight is that ΔW has rank at most r, reducing trainable parameters from dk to $(d + k)r$. For typical transformer dimensions ($d, k \approx 4096$) and modest rank ($r = 64$), this yields a compression ratio exceeding 30:1. Matrix A is initialized with scaled Gaussian values, while B starts at zero, ensuring $\Delta W = 0$ initially. During forward passes, we compute $h = W_0x + B(Ax)$ by first projecting to rank r, then back to dimension d. Only A and B receive gradients; W_0 stays frozen. At inference, $W_0 \leftarrow W_0 + BA$ merges the adaptation, eliminating overhead.

We apply LoRA with rank $r = 64$ to attention layers (query, key, value, output projections) and feed-forward networks (gate, up, down projections), updating only 1.96% of parameters across both SFT and RL training. This design preserves pre-trained visual and linguistic capabilities, reducing overfitting to synthetic CO instances and proving critical for generalization to out-of-distribution problem sizes and instance types.

C Experiment Settings

C.1 Visual Self-Evolution Training

We generate 5000 training instances per CO problem using the VAI generation process described in Section A. For supervised fine-tuning, we focus on small-scale instances: routing and graph problems use instances with fewer than 40 nodes, while scheduling problems use instances smaller than 8×8. Reference solutions are obtained using domain-specific solvers: LKH-3 for V-TSP and V-CVRP, Gurobi for V-MIS and V-MVC, CP-SAT (OR-Tools) for V-JSSP, and QIG for V-PFSP.

The visual self-evolution (VSE) stage uses an evolutionary algorithm to discover effective visualization functions for rendering problem instances. The evolution process maintains a population of code candidates (visualization functions) with population size set to 6. We use rank-based selection for parent selection and elitist preservation strategy to retain the best individual across iterations. The mutation rate is set to 0.5, meaning 3-4 new individuals are generated via mutation per iteration. The LLM backend for code generation is GPT-4o-mini with temperature 1.0. Evolution continues for a maximum of 10 function evaluations, where each evaluation involves: (1) generating images using the candidate visualization function, (2) fine-tuning a VLM on the generated images, and (3) evaluating the fine-tuned model's performance on held-out instances. Each evaluation has a timeout of 4000 seconds to accommodate the full pipeline.

We provide an example of the evolutionary search prompts used for V-CVRP visual design. The search is guided by three key components. First, the function signature specifies the interface:

Function Signature

```
def draw_cvrp_v{version}(nodes: int, demands: int, num_nodes:
int, capacity: int, out_path: str) -> None
```

Second, the task description provides design objectives and constraints:

Task Description (func_desc.txt)

Design a draw_v2 function to render CVRP instances as VLM-friendly visual representations. The image should encode routing-related priors (spatial similarity, distance, demand, clustering patterns) enabling the VLM to infer potential route groups from visual cues.
Fixed Constraints: - Use a 25×25 coordinate grid - Display x/y axes with tick marks, coordinate labels, grid lines - Image resolution: DPI=100 (fixed) - Libraries: matplotlib and numpy only (no sklearn, scipy, pandas, seaborn)
Required Visual Guidance: The visualization must apply at least one routing-aware color encoding to customer nodes. For example, - Color by angular sector: assign colors based on polar angle from depot (e.g., nodes in 0°–90° use red, 90°–180° use blue) - Color by distance: apply a gradient from near (light) to far (dark) based on Euclidean distance to depot - Color by spatial cluster: group spatially proximate nodes and assign each group a distinct color - Color by demand: map demand magnitude to color intensity (e.g., low demand in light shades, high demand in dark shades)

Third, a seed function provides the starting point for evolution:

Seed Function

```
def draw_cvrp_v0(nodes, demands, num_nodes, capacity, out_path):
    fig, ax = plt.subplots(figsize=(10, 10), dpi=100)
    ax.set_xlim(0, 25)
    ax.set_ylim(0, 25)
    ax.set_aspect('equal')

    # ... (implementation details) ...

    # Draw depot as green star
    ax.scatter(nodes[0][0], nodes[0][1], c='green',
               marker='*', s=500, label='Depot', zorder=3)

    # Draw customers as blue circles scaled by demand
    for i in range(1, num_nodes):
        size = 100
        ax.scatter(nodes[i][0], nodes[i][1], c='blue',
                   s=size, alpha=0.7, zorder=2)

    plt.savefig(out_path, bbox_inches='tight')
    plt.close()
```

The seed function uses uniform blue circles for all customer nodes and a green star marker for the depot. We also provide the mutation and crossover prompt templates below.

Mutation Prompt

{user_generator}

[Prior direction]
{direction}

[Code]
{func_signature1}
{elitist_code}

[Improved code]
Please write a mutated function {func_name}_v2, according to the direction. Strictly follow the function input and output definitions in {func_signature1}. Output code only and enclose your code with a Python code block.

Crossover Prompt

{user_generator}

[Worse code]
{func_signature0}
{worse_code}

[Better code]
{func_signature1}
{better_code}

[Direction]
{direction}

Please write an improved function {func_name}_v2, according to the search directions. Strictly follow the function input and output definitions in {func_signature1}. Output code only and enclose your code with a Python code block.

After evolution converges, the best visualization function generates images for all training instances. These images are then paired with solutions from domain solvers to create (VAI, solution) pairs for supervised fine-tuning.

C.2 Fine-tuning Training

We apply LoRA with rank $r = 64$ and scaling factor $\alpha = 64$ for parameter-efficient fine-tuning. The vision encoder remains frozen (finetune_vision_layers=False), while both attention and MLP modules in the language decoder are fine-tuned (finetune_language_layers=True, finetune_attention_modules=True, finetune_mlp_modules=True). The context length is set to 20,000 tokens to accommodate long routing solutions.

For the SFT stage, we use a per-device batch size of 4 with gradient accumulation steps of 4, yielding an effective batch size of 16. Optimization is performed using the AdamW optimizer (8-bit precision) with a learning rate of 2×10^{-4}, weight decay of 0.01, and a linear learning rate scheduler with 20 warmup steps. We train for 1 epoch on 50,000 instances per problem. The model is trained in mixed precision (bfloat16 on supported hardware, float16 otherwise) without quantization (load_in_4bit=False).

For the RL stage, we generate an additional set of at most 3,000 instances per problem. We set the hyperparameters $\beta = 0.1$ (KL penalty coefficient) and $\gamma = 0.05$ (value function coefficient). The batch size is 8, and $S = 8$ candidate solutions are sampled per instance to calculate the group advantage A_i. The weighting parameter $\lambda = 1$ balances optimality and feasibility rewards. The learning rate for RL is set to 1×10^{-6}, with other parameters matching the SFT configuration.

We structure training samples as multi-turn conversations following the standard chat format. Each sample consists of three messages: (1) a system message defining the model's role as an expert combinatorial optimization solver, (2) a user message containing the VAI (image + textual instruction

specifying problem constraints and output format), and (3) an assistant message providing the ground-truth solution. The conversation template is shown below:

Conversation Template for SFT

System: {system_prompt}
User: [Image] {instruction}
Assistant: {output}

For baseline comparisons involving OpenAI, Anthropic, and Google models, we utilize their official APIs. The experiments related to VLMs are conducted 3 times independently, and the best result is reported. All experiments are conducted on a server equipped with an Intel Xeon Platinum 8470Q CPU (22 vCPUs, 110 GB RAM) and NVIDIA RTX 6000 Ada GPUs (96 GB VRAM each) running Ubuntu 22.04, PyTorch 2.3.0, and CUDA 12.1. As a reference, the SFT process for the V-CVRP task requires approximately 44 hours of training time and consumes around 30 GB of GPU memory.

D Baseline Details

General-purpose VLMs. We evaluate six general-purpose vision-language models: OpenAI GPT-4o and GPT-4 Turbo, Anthropic Claude-Haiku-4.5 and Claude-Sonnet-5, and Qwen2.5-VL-72B and Qwen3-VL-32B. For closed-source models (GPT and Claude variants), we use their official APIs. For open-source Qwen models, we run inference locally. Each model is prompted with the same VAI (visual-attributed instance) format used in our supervised fine-tuning, consisting of the rendered image and the instruction text as shown in Section A.

Reasoning Models. Our reasoning-model baselines comprise six state-of-the-art vision-enabled models: OpenAI GPT-5.6 Sol and GPT-6-Astra, Anthropic Claude-Opus-5 and Claude-Fable-5.1, Google Gemini-2.5-Pro, and Zhipu GLM-4.5V. These models are operated in their reasoning mode with vision-enabled inference. The same VAI format (image + instruction) is applied as used for general-purpose VLMs. Reasoning models can leverage extended chain-of-thought reasoning to iteratively refine solutions, though this comes at the cost of significantly longer inference times compared to direct generation.

Prompt Strategies. We adapt three representative text-based LLM optimization methods to VLM settings and include one VLM-native baseline for comprehensive comparison. **AlgoVP** (Yang et al., 2024) iteratively searches for effective prompts by learning from prior solutions and their quality feedback; we adapt it to operate on visual instances by providing the VAI format. **VP-SGE** (Visual Self-Guiding Exploration) (Iklassov et al., 2024) leverages language models to generate multiple heuristic or metaheuristic programs in parallel, then executes them to solve the CO problem; we adapt it to VLMs by prompting the model to produce three different method names simultaneously and generate corresponding code for each method (defaulting to genetic algorithm, simulated annealing, and greedy algorithm if generation fails). **V-PHP** (Visual Progressive-Hint Prompting) (Zheng et al., 2023) scales test-time performance through iterative refinement with 4 progressive iterations, incorporating feedback based on solution feasibility and quality; we adapt it to visual inputs using the VAI format. **MVLM** (Huang et al., 2025) is a VLM-native multimodal solver that directly produces solutions from visual problem instances through prompt-guided generation. All prompt-based baselines are implemented using Claude-Sonnet-5 for fair and consistent comparison.

Domain-specific Methods. Traditional heuristics and solvers serve as the baseline group. For V-TSP, we use nearest neighbor (NN) and farthest insertion (FI) heuristics. For V-CVRP, we use the sweep algorithm and parallel savings (PS) algorithm. For V-MIS, we employ greedy minimum degree (Greedy) and degree-based add (Degree) heuristics. For V-MVC, we use the Approx method, greedy maximum degree (Greedy), and degree-based removal (Degree) heuristics. For V-PFSP, we use Palmer's and Nawaz-Enscore-Ham (NEH) (Kalczynski & Kamburowski, 2007) heuristics. For V-JSSP, we use dispatching rules including shortest processing time (SPT), first-in-first-out (FIFO), and apparent tardiness cost (ATC). We also compare against OR-Tools (Cuvelier et al., 2023) for

Table 4: Solving time (seconds) of different methods on visual CO problems.

Method	V-TSP	V-CVRP	V-MIS	V-MVC	V-PFSP	V-JSSP
General-purpose VLMs						
GPT-4o	5.5	2.8	3.9	1.6	3.1	6.0
GPT-4 Turbo	9.3	3.2	6.8	30.3	6.5	11.2
Claude-Haiku-4.5	4.0	3.1	11.5	3.6	3.6	3.9
Claude-Sonnet-5	10.4	30.7	36.9	19.3	19.8	13.1
Qwen2.5-VL-72B	2.9	0.9	3.3	1.0	1.5	2.2
Qwen3-VL-32B	4.7	0.7	4.1	1.6	1.2	1.8
Reasoning Models						
GPT-5.6 Sol	29.1	49.3	47.2	31.2	67.0	47.8
GPT-6-Astra	33.1	59.2	47.6	21.5	55.9	37.4
Claude-Opus-5	20.6	16.5	40.4	9.3	39.4	4.7
Claude-Fable-5.1	6.7	19.3	27.4	5.1	34.0	5.6
Gemini-2.5-Pro	85.7	78.4	43.2	69.8	41.5	93.2
GLM-4.5V	61.3	85.6	66.5	22.9	43.9	110.0
Prompt Strategies						
AlgoVP	64.4	54.8	247.4	82.8	160.5	97.2
VP-SGE	110.6	56.1	101.7	47.2	91.1	62.2
VP-PHP	120.8	43.2	108.6	45.9	77.4	79.1
MVLM	88.4	91.2	135.7	74.3	66.8	101.5
End-to-End Strategies						
Ours (SFT)	5.4	4.9	2.8	3.5	1.7	9.2
Ours (SFT+RL)	5.4	4.9	2.8	3.5	1.7	9.2
Ours (SFT+RL+BoN)	9.4	8.6	3.8	6.3	2.5	15.1
Ours (SFT+RL+VA)	10.1	9.4	4.5	7.1	2.8	16.6

routing problems and ant colony optimization (ACO) as a metaheuristic baseline. All domain-specific methods operate on the numerical data extracted from visual instances.

E Additional Experiment Results

E.1 Detailed Running Time

We report per-instance solving times across six visual CO problems in Table 4. Our fine-tuned 7B VLM achieves a favorable speed-quality trade-off, running orders of magnitude faster than reasoning models while delivering competitive solution quality. Timing differences among proprietary APIs reflect variations in underlying infrastructure and are not directly comparable.

E.2 Example Reasoning Process

To illustrate the reasoning capabilities of state-of-the-art vision-enabled reasoning models on visual CO problems, we examine GPT-6-Astra's chain-of-thought (CoT) generation for a V-CVRP instance with 45 customers. The model systematically extracts visual information and applies heuristic procedures to construct a solution. Below we present a condensed excerpt that captures the key reasoning steps while preserving the model's deliberative problem-solving approach.

GPT-6-Astra CoT for a V-CVRP Instance (1/2)

I'm transcribing the plotted coordinates and demands first, then I'll run the savings procedure against those exact grid points and verify the resulting partition and distance. The labels are dense in a few regions, so I'm checking the image systematically before calculating merges.The plot uses integer grid coordinates, so Euclidean distances can be computed directly from the plotted positions. I'm resolving the overlapping labels around the right-middle and lower-right points before producing the savings order; those IDs affect both the route names and the merge sequence.Customer demands: [C1=9, C2=3, C3=9, C4=7, C5=4, C6=3, C7=10, C8=5, C9=5, C10=3, C11=9, C12=4, C13=5, C14=4, C15=5, C16=7, C17=9, C18=7, C19=5, C20=10, C21=10, C22=7, C23=10, C24=5, C25=3, C26=7, C27=7, C28=9, C29=10, C30=10, C31=11, C32=6, C33=7, C34=8, C35=10, C36=9, C37=2, C38=10, C39=2, C40=5, C41=5, C42=6, C43=3, C44=8, C45=1]
Total customers: 45. Depot: node 0. Vehicle capacity: 111.
The plotted coordinates were read on the 25 x 25 grid. The depot is at '(10,17)'. Each customer initially formed a dedicated route '[0, customer, 0]'.
For every customer pair, I computed:
'Saving(i,j) = distance(0,i) + distance(0,j) - distance(i,j)'
The savings were sorted in descending order. Pairs were accepted only when both customers were route endpoints and the combined load was at most 111. The accepted merges, in descending savings order, were:
Merges performed: '(6,35), (35,20), (20,24), (24,25), (25,5), (5,14), (14,7), (7,12), (12,17), (17,15), (15,1)' '(19,27), (27,22), (22,21), (21,4), (4,40), (40,16), (16,42), (42,26), (26,30), (30,31), (31,33), (33,13)' '(2,10), (10,3), (3,8), (8,45), (45,9), (9,28), (28,23), (23,38), (38,11), (11,34)' '(43,18), (18,37), (37,41), (41,39), (39,32), (32,36), (36,44), (44,29)'
All remaining savings were skipped because the selected customers were no longer both route endpoints or the merge would have produced an invalid route structure.

GPT-6-Astra CoT for a V-CVRP Instance (2/2)

Per-route load:
- Route 1: '[0, 6, 35, 20, 24, 25, 5, 14, 7, 12, 17, 15, 1, 0]', load '76' - Route 2: '[0, 19, 27, 22, 21, 4, 40, 16, 42, 26, 30, 31, 33, 13, 0]', load '94' - Route 3: '[0, 2, 10, 3, 8, 45, 9, 28, 23, 38, 11, 34, 0]', load '69' - Route 4: '[0, 43, 18, 37, 41, 39, 32, 36, 44, 29, 0]', load '52'
Every customer appears exactly once, and every route load is at most 111.
Final routes: '[[0, 6, 35, 20, 24, 25, 5, 14, 7, 12, 17, 15, 1, 0], [0, 19, 27, 22, 21, 4, 40, 16, 42, 26, 30, 31, 33, 13, 0], [0, 2, 10, 3, 8, 45, 9, 28, 23, 38, 11, 34, 0], [0, 43, 18, 37, 41, 39, 32, 36, 44, 29, 0]]'
Route distances:
- Route 1: approximately '56.56' - Route 2: approximately '55.49' - Route 3: approximately '71.36' - Route 4: approximately '48.64'
Total distance: approximately '232.05'
Routes: [[0, 6, 35, 20, 24, 25, 5, 14, 7, 12, 17, 15, 1, 0], [0, 19, 27, 22, 21, 4, 40, 16, 42, 26, 30, 31, 33, 13, 0], [0, 2, 10, 3, 8, 45, 9, 28, 23, 38, 11, 34, 0], [0, 43, 18, 37, 41, 39, 32, 36, 44, 29, 0]], Objective: 232.05

Table 5: Ablation on visual encoder fine-tuning: comparing configurations on CVRP and JSSP.

Configuration	**CVRP**				**JSSP**			
	SFT		Full		SFT		Full	
	fea.	opt.	fea.	opt.	fea.	opt.	fea.	opt.
Fine-tuned vision encoder	31%	36.84%±40.7	55%	28.21%±2.9	42%	26.45%±22.7	68%	31.60%±32.3
Frozen vision encoder	62%	14.30%±9.9	100%	9.07%±6.3	78%	18.43%±13.5	100%	10.93%±8.9

This example shows GPT-6-Astra successfully extracts visual information, applies Clarke-Wright savings with capacity checks, and produces feasible routes. However, the greedy merge strategy leads to a significant optimality gap by accepting savings in descending order without considering downstream effects. While reasoning models can implement classical algorithms on visual instances, high-quality solutions require specialized fine-tuning to learn adaptive optimization patterns beyond generic execution.

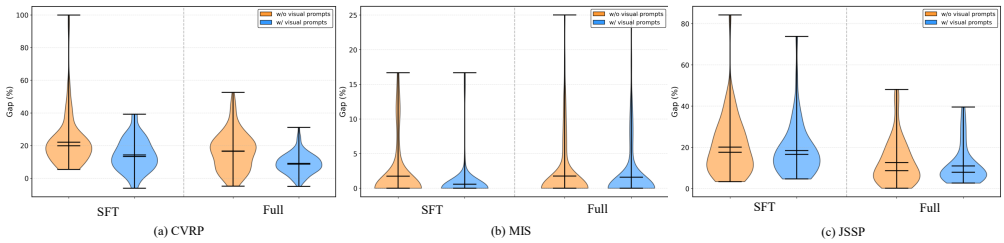


Figure 10: Distribution of constraint margins on SFT and Full.

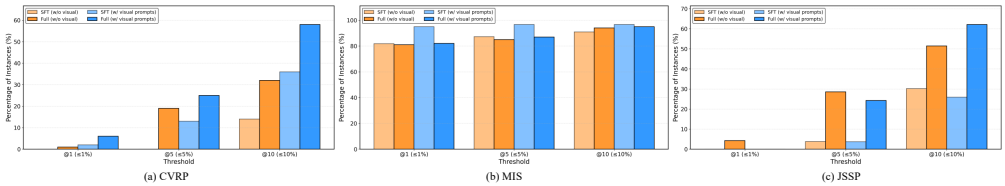


Figure 11: Performance in Gap@K.

E.3 Ablation on Encoder Fine-tuning

To assess the contribution of fine-tuning the visual encoder, we compare freezing versus fine-tuning the vision encoder while training the language decoder. We evaluate on CVRP and JSSP across two training stages: supervised fine-tuning alone (denoted SFT) and the complete pipeline incorporating reinforcement learning (denoted Full). As shown in Table 5, freezing the vision encoder consistently achieves higher feasibility rates and lower optimality gaps across both problems and training stages. This indicates that fine-tuning the vision encoder may lead to overfitting on the training distribution. These results suggest that pre-trained vision representations generalize better than adapted ones for CO visual patterns.

Table 6: Ablation study on visual prompts.

Configuration	**V-CVRP**		**V-MIS**		**V-JSSP**	
	fea.	opt.	fea.	opt.	fea.	opt.
SFT w/o visual prompts	43%	22.13%±15.2	55%	0.88%±2.4	71%	20.08%±14.3
SFT w/ visual prompts	62%	14.30%±9.9	60%	0.82%±2.2	78%	18.49%±13.5
Full w/o visual prompts	83%	16.54%±11.8	100%	2.23%±4.2	100%	12.63%±10.0
Full w/ visual prompts	100%	9.07%±6.3	100%	2.08%±4.3	100%	10.93%±8.9

E.4 Ablation on Visual Prompts

We incorporate visual prompts, such as spatial clustering coloring and degree-based shading, into the rendered problem images to guide the model toward promising solution regions. These visual cues serve a role similar to feature engineering, providing the VLM with heuristic hints that facilitate more effective solution generation. For example, color-coded routes can bias the model toward spatially coherent clusters, while node shading can prioritize low-degree nodes in graph problems. To assess the contribution of visual prompts, we conduct an ablation study comparing models with and without visual prompts.

Table 6 shows that removing visual prompts leads to a performance drop across three representative problems (CVRP, MIS, JSSP). Figure 10 illustrates the distribution of constraint margins, demonstrating that visual prompts help both SFT and Full models achieve tighter constraint satisfaction. Figure 11 further shows the percentage of instances achieving gaps below 1%, 5%, and 10%, where models with visual prompts consistently outperform those without across both training stages. These findings indicate that visual prompts effectively guide the VLM's solution-space exploration during both training and inference.

E.5 Generalization across VLM Backbones

To further demonstrate the versatility of our proposed framework, we fine-tune two other widely adopted open-source VLMs, Qwen3-VL-8B and InternVL3-8B, as visual combinatorial optimization solvers. The fine-tuning configurations are the same as the Qwen2.5-VL-7B model. As shown in Table 7, all three VLMs achieve competitive performance across the six CO tasks, with Full configurations consistently reaching high feasibility and low optimality gaps. Despite the different model architectures and training schemes, the proposed method consistently maintains its effectiveness across all CO tasks and VLM backbones.

Table 7: Comparison between Qwen2.5-VL-7B, Qwen3-VL-8B, and InternVL3-8B on the studied CO problems.

Problem	Qwen2.5-VL-7B				Qwen3-VL-8B				InternVL3-8B			
	SFT		Full		SFT		Full		SFT		Full	
	fea.	opt. (%)	fea.	opt. (%)	fea.	opt. (%)	fea.	opt. (%)	fea.	opt. (%)	fea.	opt. (%)
TSP	69%	8.32%±5.6	100%	5.97%±4.1	78%	8.57%±5.3	100%	6.51%±4.7	75%	8.06%±3.7	100%	6.03%±3.9
CVRP	62%	14.30%±9.9	100%	9.07%±6.3	65%	13.52%±8.3	100%	8.63%±6.1	65%	15.07%±11.6	100%	9.38%±6.8
MIS	60%	0.82%±2.2	100%	2.08%±4.3	65%	0.84%±2.5	100%	2.03%±4.0	70%	1.39%±2.2	92%	1.95%±4.1
MVC	75%	1.89%±3.2	100%	1.98%±3.1	69%	1.85%±3.2	100%	2.16%±3.2	73%	1.56%±3.3	100%	1.83%±2.5
PFSP	100%	4.53%±3.1	100%	1.85%±2.5	100%	3.36%±2.8	100%	1.98%±2.4	100%	4.39%±3.6	100%	2.24%±2.9
JSSP	78%	18.43%±12.9	100%	10.93%±8.9	73%	15.78%±10.3	100%	9.03%±8.0	66%	15.93%±13.5	96%	9.17%±8.8

E.6 Unified CO Solver

To demonstrate the potential of our approach in enabling a unified CO solver, we fine-tune a single Qwen2.5-VL-7B model using the combined training datasets from all six studied CO problems via SFT. As a baseline, we also evaluate a text-based unified CO solver that operates on textual problem descriptions instead of visual inputs. For fair comparison, both models are evaluated with $N = 8$ sampling, and the results are reported in Table 8. The results show that while text-based solver achieves higher feasibility rates and tighter optimality gaps on most problems, our vision-based unified solver demonstrates competitive performance across all tasks without accessing explicit numerical problem data. These results suggest that training across multiple problem types using visual representations does not significantly compromise the model's ability to generate high-quality solutions. With further scaling in model size, diversity of CO tasks, and visual prompt design, VLMs hold significant promise as foundation models for combinatorial optimization.

Table 8: The performance of the unified CO solver.

Method	V-TSP		V-CVRP		V-MIS		V-MVC		V-PFSP		V-JSSP	
	fea.	opt.	fea.	opt.	fea.	opt.	fea.	opt.	fea.	opt.	fea.	opt.
Text-based solver	100%	0.86%	93%	4.50%	84%	1.89%	93%	2.47%	100%	1.31%	100%	8.25%
Our	92%	6.52%	85%	9.28%	86%	2.16%	100%	2.60%	100%	1.94%	91%	13.54%

E.7 Out-of-distribution Performance

To evaluate the out-of-distribution (OOD) performance of the trained models, we adopt three OOD distributions commonly used as benchmarks for routing problems in prior work. The uniform distribution samples node coordinates uniformly from $\mathcal{U}[1, 1000]^2$. The Gaussian distribution samples coordinates from a Gaussian mixture with parameters $(c = 2, l = 5)$. The clustered distribution generates city nodes by first sampling 7 centroids uniformly at random, followed by sampling node locations from a normal distribution centered at each centroid with a standard deviation of 0.1. The evaluation results on V-TSP and V-CVRP are shown in Table 9. According to the table, we can find that there is a slight performance drop when evaluating on the OOD datasets. Since the drop is not significant, the cross-distribution generalizability of our models is proven.

Table 9: Comparative evaluations of feasibility rate (fea.) and optimality gap (opt.) between in-distribution (InD) and out-of-distribution (OOD) data.

Problem	Method	InD		OOD (Uniform)		OOD (Gaussian)		OOD (Clustered)	
		fea.	opt.	fea.	opt.	fea.	opt.	fea.	opt.
V-TSP	Ours (BoN)	100%	5.97%	100%	7.21%	100%	9.03%	100%	5.83%
	Ours (VA)	100%	5.69%	100%	7.06%	100%	8.68%	100%	5.57%
V-CVRP	Ours (BoN)	100%	9.07%	100%	9.75%	100%	8.74%	100%	10.71%
	Ours (VA)	100%	8.50%	100%	9.38%	100%	8.79%	98%	10.66%

E.8 Benchmarking Performance of V-TSP

To further demonstrate the generalizability of our method, we evaluate the V-TSP model on a benchmark dataset. Specifically, we use representative instances with fewer than 100 nodes from TSPLIB Zhu et al. (2026), which are derived from real-world routing scenarios. We compare our results with traditional heuristics (NN, FI, ACO) and text-based solvers. The optimality gaps are presented in Table 10. Our method achieves competitive performance on these benchmark instances, indicating it can generalize to real-world scenarios.

Table 10: Optimality gap comparison of VLM-based methods on TSPLIB instances.

Instance	Size	NN	FI	ACO	Text-based	Ours (BoN)	Ours (VA)
eil51.tsp	51	20.57%	5.06%	11.27%	2.02%	4.59%	3.98%
berlin52.tsp	52	19.08%	7.67%	3.06%	4.54%	4.34%	4.62%
st70.tsp	70	19.34%	10.04%	5.76%	0%	5.51%	5.93%
eil76.tsp	76	32.09%	8.17%	8.71%	2.91%	8.66%	7.71%
pr76.tsp	76	40.74%	9.73%	10.62%	3.87%	9.03%	8.48%
Average Gap		26.36%	8.13%	7.88%	2.67%	6.43%	6.14%

Similarly, we evaluate our V-JSSP model on standard benchmark instances from Taillard's JSSP benchmark suite Graviers et al. (2025). We compare our method against classical dispatching rules (FIFO, ATC), learned construction heuristics (L2D). Table 11 shows the optimality gap on representative instances. Our visual-based approach demonstrates competitive performance across different problem scales.

Table 11: Optimality gap comparison on Taillard JSSP benchmark instances.

Instance	Size	FIFO	ATC	L2D	Text-based	Ours (BoN)	Ours (VA)
ta01.jssp	15×15	34.87%	29.12%	23.88%	14.22%	18.50%	18.23%
ta11.jssp	20×15	47.15%	31.46%	32.86%	21.16%	27.81%	27.69%
ta21.jssp	20×20	42.03%	30.72%	31.21%	23.30%	32.29%	32.54%
Average Gap		41.35%	30.43%	29.32%	19.56%	26.20%	26.15%