GOBench: Stage-Wise Diagnostics and the Visual Paradox in Multimodal Graph Optimization

Yinghao Chen[1]*, **Wantong Xie**[2]*, **Shuli Zeng**[1]† , **Sijia Zhang**[1], **Xiaotian Pan**[1]
Feng Wu[1] and **Xiang-Yang Li**[1]
[1]School of Computer Science and Technology, University of Science and Technology of China
[2]Institute of Advanced Technology, University of Science and Technology of China
{chenyinghao,wantongxie,zengshuli0130,sxzsj,px259637522}@mail.ustc.edu.cn
{wufeng02,xiangyangli}@ustc.edu.cn

Abstract

Large language models (LLMs) and vision-language models (VLMs) are increasingly used as optimization assistants to produce solutions, generate solver-executable programs, or both. However, current evaluations are misaligned with deployment in three ways: they (P1) fail to represent multimodal problem specifications, (P2) score outcomes only and cannot localize where failures occur along the modeling pipeline, and (P3) rarely report inference cost, obscuring reliability–cost trade-offs. We introduce Graph Optimization benchmark (**GOBench**), an aligned multimodal benchmark with solver-derived oracles and a four-layer diagnostic protocol that evaluates intermediate artifacts as well as end results, together with the **Visual Inference Penalty (VIP)** to measure multimodal overhead. Across frontier and open-weight models under paired text-only vs. T+V settings, we find that vision reliably increases inference cost, while its reliability impact is regime-dependent: frontier models often benefit from visual grounding, whereas several mid-tier/open models exhibit a **Visual Paradox** where vision reduces downstream executability and verification coverage. End-to-end success is frequently bottlenecked by intermediate-stage dropout; supervised fine-tuning on intermediate targets can mitigate this attrition in open models, enabling a reproducible harness for diagnosing failure modes and quantifying reliability–cost trade-offs.

1 Introduction

Motivation. In practice, combinatorial optimization problems are often specified in multiple modalities: diagrams convey graph structure, while constraints and objectives are described in text (Mirhoseini et al., 2021; Zhang et al., 2024). This raises a practical question for optimization assistants: when does adding a diagram help beyond text-only input, and when can it hurt? Answering this requires evaluation on aligned text–diagram views of the same instance, with diagnostics that localize failures along the modeling pipeline and with explicit reporting of the cost of enabling vision (Fu et al., 2025; Zeng et al., 2024).

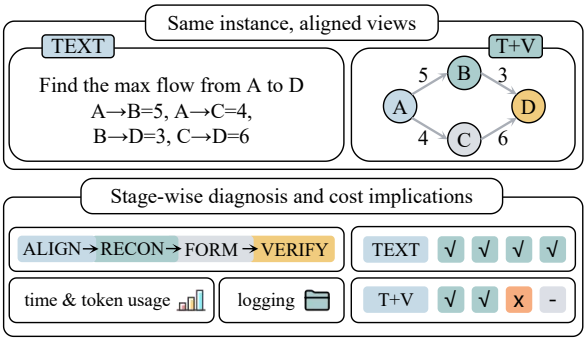


Figure 1: **aligned paired views reveal stage-wise multimodal failure.** GOBench evaluates the same graph-optimization instance under aligned TEXT and T+V views, enabling controlled paired comparison. Rather than scoring only final outcomes, our protocol localizes failures across ALIGN, RECON, FORM, and VERIFY, while also logging inference cost. In some models, adding vision increases cost yet shifts failure earlier in the pipeline, illustrating the Visual Paradox.

Current benchmarks remain misaligned with deployment in three ways. **(P1) Modality realism:** Existing benchmarks such as NL4Opt (Ramamonjison et al., 2023) are text-only, making them missing diagram-grounded specifications. **(P2) Process diagnostics:** focus solely on outcome cannot localize failures along the modeling pipeline, which obscures the root causes of errors (Turpin et al., 2023). **(P3) Deployment cost:** Inference cost is rarely reported (Liang et al., 2023), making reliability–cost trade-offs difficult to assess (Chen et al., 2024).

We introduce **GOBench**, an aligned multi-view benchmark and evaluation harness for graph optimization. Starting from a structured seed, we deter-

*These authors contributed equally to this work.
†Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2026, pages 6144–6167

Table 1: **Comparison with existing optimization-assistant benchmarks.** We highlight whether a benchmark supports process diagnostic, reports an explicit **cost metric**, and covers **multimodal** specifications.

Benchmark	Multi-Modal	Outcome Metric	Process Diagnostic	Cost Metric
NL4Opt (Ramamonjison et al., 2023)	×	✓	×	×
OptiGuide (Li et al., 2023)	×	✓	×	×
MAMO (Huang et al., 2025b)	✓	✓	×	×
IndustryOR (Huang et al., 2025a)	×	✓	×	×
GOBench (Ours)	✓	✓	✓ (4-layer)	✓

ministically generate a textual view and a Graphviz-rendered diagram of the same instance, together with solver-derived oracles. Inspired by the multilevel lens of Teng (1999), we treat modeling as a staged process and operationalize it with a four-layer diagnostic protocol: ALIGN, RECON, FORM, and VERIFY. We additionally log inference cost and report the **VIP** to quantify multimodal overhead.

Across frontier and open-weight models under paired text-only vs. T+V settings, we observe frequent intermediate-stage dropout that dominates end-to-end success. We also find a **Visual Paradox**: for several non-frontier models, diagrams increase cost yet reduce reliability, especially by destabilizing reconstruction and formulation; in contrast, frontier models more often benefit from visual grounding.

Our contributions are summarized as follows:

- **GOBench: an aligned multi-view benchmark with solver oracles.** We introduce **GOBench**, a multimodal graph-optimization benchmark that provides paired TEXT and T+V views of the same instance under strict information alignment, together with solver-derived oracle solutions for ground-truth verification.
- **A four-layer diagnostic protocol with clean failure attribution.** We propose a staged evaluation protocol—ALIGN, RECON, FORM, and VERIFY—implemented by deterministic validators and solver execution, enabling stage-wise reliability analysis and earliest-failure attribution of pipeline dropouts.
- **Cost-aware multimodal evaluation via VIP and paired analysis.** We log tokens and latency for every instance and define the VIP to quantify multimodal overhead, enabling controlled paired comparisons that reveal reliability–cost trade-offs and the Visual Paradox regime where vision increases cost yet degrades executability.

2 Related Work

Benchmarking Optimization Assistants. Prior work evaluates LLMs for optimization mainly via text-to-optimization translation (Ramamonjison et al., 2023; Huang et al., 2025a) and solver-in-the-loop pipelines (Peng et al., 2018), typically emphasizing final outcomes. NL4Opt (Ramamonjison et al., 2023) pioneered evaluation on linear programming word problems, but it is text-only and does not reflect diagram-grounded specifications that appear in technical documents (Mathew et al., 2021). More recent benchmarks such as MAMO (Huang et al., 2025b) and IndustryOR (Huang et al., 2025a) introduce richer constraints, yet they still primarily report final correctness or executability, leaving intermediate failures unobserved (Yang et al., 2023).

Process-Grounded Evaluation and Intermediate Supervision. Outcome-only metrics can obscure where errors arise and encourage unfaithful intermediate reasoning (Dziri et al., 2023; Turpin et al., 2023). Related lines of work study interpretability or tool-assisted optimization, but they often assume a correct mathematical model is already available rather than requiring end-to-end modeling from raw inputs (Creswell and Shanahan, 2022).

Multimodal Extraction from Diagrams. Recent work on extracting structured data from visual artifacts shows that visual grounding can be a bottleneck for downstream reasoning (Liu et al., 2023), and document understanding benchmarks highlight the difficulty of mapping text to complex layouts (Mathew et al., 2021).

Positioning of GOBench. Table 1 summarizes how GOBench differs from representative optimization-assistant benchmarks. Our novelty is not a new optimization model or solver architecture, but an evaluation design that combines three elements that are usually studied separately: paired-view control, solver-grounded stage diagnostics,

and cost-aware deployability. Concretely, GOBench evaluates the same instance under aligned TEXT and T+V views, localizes failures across a four-layer protocol, and reports multimodal overhead through VIP. This design makes it possible to distinguish whether multimodal gains or failures arise from visual grounding, symbolic projection, or final solution quality, while also exposing cases where enabling vision increases cost without improving end-to-end verification.

3 GOBench: Dataset and Protocol

3.1 Aligned Multi-View Synthesis

GOBench is built from a controlled synthesis pipeline that enforces strict information alignment across three views: a structured seed s, a textual view t, and a diagram view v (Figure 2). We generate diverse graphs using `NetworkX` (Hagberg et al., 2007), varying scale $|V| \in [5, 20]$ and density $\rho \in [0.1, 0.5]$, while ensuring feasibility for each task instance. Given s, we render t via surface realization and v via Graphviz (Gansner and North, 2000). We additionally compile s into a canonical solver format and compute solver-derived oracles, which provide a ground-truth objective value and reference assignments used by the VERIFY layer. This aligned generation makes paired evaluation fair: any difference between text-only and T+V runs cannot be attributed to missing or extra information, only to the model's ability to use the diagram.

3.2 Task Families

GOBench is task-agnostic at the level of data synthesis and diagnostics: the same pipeline generates aligned views and the same four-layer protocol evaluates intermediate artifacts, while the oracle, formulation, and verification logic are task-specific.

Maximum Flow. We instantiate GOBench on a directed capacitated max-flow family. The structured seed specifies (V, E, c) and a source–sink query. The oracle is obtained by solving the corresponding flow program, and verification checks feasibility and objective quality against the oracle.

Shortest Path. We additionally instantiate the same framework on shortest path. The structured seed and aligned text/diagram views follow the same generation pipeline, but the oracle and verification are replaced with shortest-path solvers and validators, and the executable formulation encodes a shortest-path objective. This extension demonstrates that the benchmark design and diagnostics are not tied to a single problem type.

3.3 Four-Layer Diagnostic Protocol

We evaluate optimization assistants as verifiable pipelines rather than black-box predictors. Inspired by the multilevel lens of Teng (1999), we decompose modeling into four layers that emit structured artifacts under a strict output contract. The key idea is a coarse-to-fine prerequisite decomposition: ALIGN establishes the canonical entity binding, RECON recovers the normalized instance under that binding, FORM projects the reconstructed instance into a solver-executable program, and VERIFY evaluates solution quality only once the upstream artifacts are valid. Figure 3 provides an end-to-end overview.

Layer 1 (ALIGN): anchored entity binding. The model must map entities in the input to a canonical contiguous index set. We anchor this mapping to control for permutation invariance, enabling a clean paired causal comparison and preventing harmless renaming errors from confounding diagnosis of RECON/FORM/VERIFY. Failures indicate missing/confused entities or violation of the binding contract, often triggered by visual clutter or inconsistent mentions.

Layer 2 (RECON): instance reconstruction. The model must recover the full problem instance under the anchored mapping. Failures indicate topology hallucinations or extraction errors, separated from downstream coding mistakes.

Layer 3 (FORM): executable formulation. The model must generate a solver-executable program consistent with its reconstructed instance. Failures indicate syntax errors or symbolic inconsistencies that prevent compilation or correct execution.

Layer 4 (VERIFY): solver-grounded verification. We run an external solver to check feasibility and measure objective quality against the oracle. This layer isolates optimization/solution quality from earlier pipeline failures.

3.4 Cost Logging and VIP

To quantify deployment overhead, we log token usage, end-to-end latency, and solver time for every instance. We define the VIP for model m as the multiplicative cost ratio between text-only and T+V,

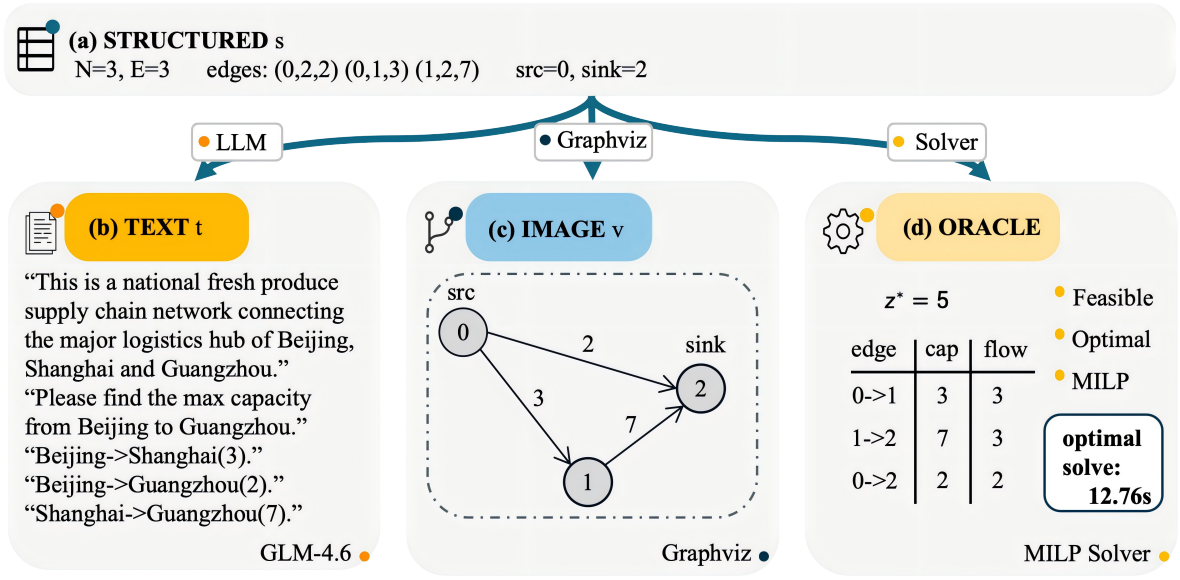


Figure 2: **Aligned multi-view synthesis in GOBench.** Starting from a structured seed graph s, we deterministically generate two isomorphic views: a textual description t and a Graphviz-rendered diagram v. The same seed is also compiled to a solver instance to obtain an oracle solution. Because t and v encode the same underlying instance θ, paired text-only vs. T+V evaluation isolates the causal effect of the visual modality. City names are randomly sampled by LLM.

to summarize the overhead introduced by enabling vision:

$$\mathrm{VIP}_m = \frac{\mathcal{C}(m, \text{T+V})}{\mathcal{C}(m, \text{TEXT-ONLY})}, \tag{1}$$

where $\mathcal{C}$ can be instantiated as tokens or latency. Reporting VIP alongside stage-wise reliability enables cost-aware comparisons and reveals when visual inputs increase cost without improving end-to-end verification.

4 Experimental Setup

We evaluate GOBench with a paired-modality protocol. For each model m, we run the same test instances under TEXT and T+V, keeping prompts and decoding fixed and varying only whether the aligned diagram is provided. Models without vision support are reported in TEXT only (no VIP).

4.1 Tasks, Data Generation, and Splits

All instances are procedurally synthesized from structured seeds using the aligned multi-view pipeline in Section 3. Unless otherwise noted, main-text results focus on **Maximum Flow**; **Shortest Path** follows the same contract and harness with task-specific oracle and verification. Graphs are generated with $|V| \in [5, 20]$ and density $\rho \in [0.1, 0.5]$. We use an 800/100/100 split for Train/Dev/Test and report all aggregate numbers on the held-out test set.

4.2 Evaluated Models and Capability Tiers

We evaluate proprietary frontier systems, strong API baselines, and open-weight models to study capability-dependent modality effects. **Tier 1** includes state-of-the-art proprietary models and serves as an empirical upper bound. **Tier 2** includes widely used high-performing API models with favorable deployment profiles. **Tier 3** includes efficient API models and open-weight checkpoints that can be locally served and fine-tuned. For open-weight experiments, we keep the serving stack consistent and report the same token/latency fields (Appendix A).

4.3 Prompt, Output Contract, and Evaluation Harness

All models use a unified instruction template (Appendix F) that enforces a strict structured output contract. We adopt a machine-parseable (NDJSON) contract to match realistic deployment settings where model outputs must be directly consumed by downstream tooling rather than read by humans. In such pipelines, free-form explanations are not reliably executable and can break automated parsing, so enforcing a minimal structured interface is a practical requirement—not an artificial handi-

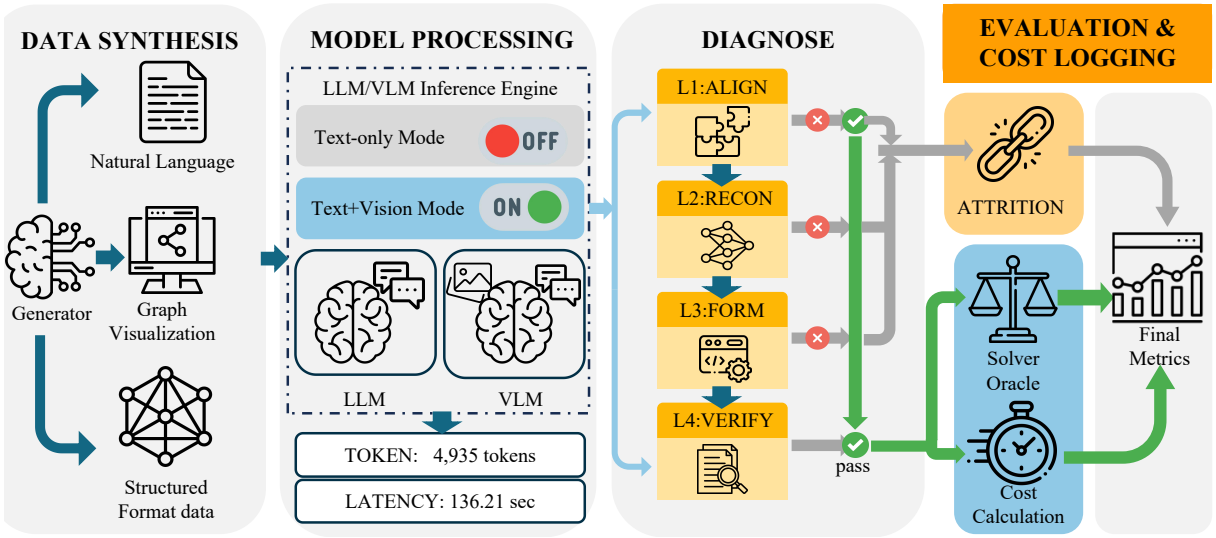


Figure 3: **Overview of the GOBench evaluation pipeline.** From aligned inputs, models must produce four artifacts, which are checked by deterministic validators and solver execution; the harness also logs token/latency to compute VIP.

cap.

For each instance, the pipeline should emit four machine-parseable artifacts: an entity mapping, a reconstructed graph, a solver formulation, and a direct solution proposal. The harness parses each artifact, applies schema checks, runs deterministic validators, and executes solver-grounded checks when applicable. Failures are treated as attrition along the pipeline: if a layer is missing or invalid, downstream layers are not credited. We also log and report `parse_violations` as first-class hard errors for deployable executability, since any non-conforming output would fail in a solver-executable pipeline (Appendix E). For completed requests, we operationalize truncation at the harness level: when generation reaches the configured `max_output_tokens` cap, the model typically returns an incomplete NDJSON fragment, which is captured as stage-wise `parse_violations` under our strict output contract.

4.4 Metrics Reported in Table 2

Let $\mathcal{D} = \{1, \ldots, N\}$ denote the test set. For a model m under mode μ, the harness outputs binary validity indicators per instance $i \in \mathcal{D}$: a_i, r_i, f_i, and v_i. Stage-wise reliability is reported as percentages:

$$S_{\text{ALIGN}}(m, \mu) = \frac{1}{N} \sum_{i \in \mathcal{D}} a_i, \tag{2}$$

$$S_{\text{RECON}}(m, \mu) = \frac{1}{N} \sum_{i \in \mathcal{D}} r_i, \tag{3}$$

$$S_{\text{FORM}}(m, \mu) = \frac{1}{N} \sum_{i \in \mathcal{D}} f_i, \tag{4}$$

$$S_{\text{VERIFY}}(m, \mu) = \frac{1}{N} \sum_{i \in \mathcal{D}} v_i. \tag{5}$$

Here $a_i = 1$ iff the mapping artifact is schema-valid; $r_i = 1$ iff the reconstructed graph is schema-valid and consistent under the mapping; $f_i = 1$ iff the solver formulation is executable; and $v_i = 1$ iff the predicted solution is feasible under the task constraints.

Objective gap. Let $z_i^\star$ be the oracle objective and $\hat{z}_i$ be the objective implied by the model prediction (computed by the harness). We define the per-instance normalized absolute objective gap:

$$g_i = \frac{|\hat{z}_i - z_i^\star|}{\max(10^{-9}, |z_i^\star|)}. \tag{6}$$

In Table 2, we report the mean gap on the verified subset:

$$\textbf{Gap}(m, \mu) = \frac{1}{\sum_{i \in \mathcal{D}} v_i} \sum_{i \in \mathcal{D}} v_i \, g_i. \tag{7}$$

Because the objective gap is only defined for solver-verified instances, Gap should be interpreted as

conditional quality and must be read together with VERIFY coverage in Table 2. A small Gap is not meaningful if $\sum_{i\in\mathcal{D}} v_i$ is small.

Cost metrics. For each instance i, let T_i denote backend-reported total tokens and let L_i denote end-to-end wall-clock latency. We report their means:

$$\textbf{Tok}(m,\mu) = \frac{1}{N}\sum_{i\in\mathcal{D}} T_i,\ \textbf{Lat}(m,\mu) = \frac{1}{N}\sum_{i\in\mathcal{D}} L_i. \tag{8}$$

VIP. For models that support both modes, VIP quantifies the multiplicative overhead of enabling vision:

$$\text{VIP}_{\text{tok}}(m) = \frac{Tok(m,\text{T+V})}{Tok(m,\text{TEXT})}, \tag{9}$$

$$\text{VIP}_{\text{lat}}(m) = \frac{Lat(m,\text{T+V})}{Lat(m,\text{TEXT})}. \tag{10}$$

We report VIP only when both modalities are available; values below 1 can occur due to serving-side variability, so token-based VIP is the more stable proxy for multimodal overhead.

5 Results and Analysis

We report quantitative results on the held-out test split using the metrics defined in Section 4.4. Unless stated otherwise, main-text results focus on the **Maximum Flow** task, where all models are evaluated under TEXT and T+V using aligned instances (Section 3.1). Models that do not support vision are reported with TEXT-only rows and have no VIP entries.

5.1 Overall performance

Table 2 summarizes end-to-end behavior together with intermediate reliability and deployment cost, directly addressing the three deployment gaps raised in Section 1: modality realism (P1), process diagnostics (P2), and cost transparency (P3). Frontier models maintain high survival through the full pipeline and achieve low objective gaps, while several mid-tier and open models exhibit sharp attrition before verification. This makes clear why outcome-only reporting is insufficient: many failures are not "bad optimization" at the final step, but invalid intermediate artifacts that prevent the solver-grounded check from being reached.

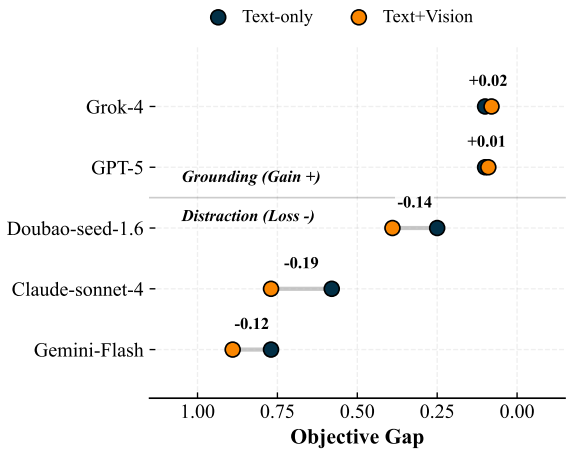


Figure 4: **Paired modality effect (text-only vs. T+V).** Frontier models show small but consistent gains in objective gap, while several mid-tier models degrade, despite higher multimodal token cost.

5.2 End-to-end success Bottleneck(P2)

A consistent pattern across model families is that end-to-end success is governed by whether intermediate artifacts survive, rather than by solver difficulty alone. In Table 2, frontier systems keep high pass rates from ALIGN through VERIFY, whereas open models often drop at RECON or FORM. This stage-wise view localizes the root cause: failures concentrate on reconstructing the instance or producing an executable formulation, not on the final verification procedure. As a result, the mean gap is meaningful only together with verification coverage, since it is computed on the verified subset, see Eq. 7.

5.3 Modality effects reverse under vision(P1)

We isolate the causal effect of adding the aligned diagram view by comparing TEXT and T+V for models that support both modes. Figure 4 shows a regime split: frontier models often benefit from visual grounding, while several mid-tier/open models exhibit a Visual Paradox in which diagrams increase cost yet degrade downstream executability and verification coverage. Table 2 clarifies where this reversal happens: for some open models, ALIGN remains high under T+V but RECON and FORM collapse, implying that visual inputs destabilize topology recovery and symbolic projection rather than entity binding alone.

5.4 Trade-offs and VIP(P3)

Figure 5 summarizes joint VERIFY coverage, verified quality, and deployment cost. Because wall-

Table 2: **Main results on GOBench (test, $n = 100$).** Stage-wise reliability reports the fraction of instances producing valid artifacts: ALIGN, RECON, FORM, VERIFY. **Tok** denotes mean total tokens. **VIP** reports the multiplicative cost of enabling vision, measured both in tokens and latency; values < 1 can occur due to backend variability. Gap is computed only on solver-verified instances (Eq. 7) and should be interpreted jointly with VERIFY coverage. Uncertainty estimates for conditional metrics are reported in Appendix E (Table 7).

Model	Mode	ALIGN	RECON	FORM	VERIFY	Gap↓	Tok	Lat(s)	VIP_{tok}	VIP_{lat}
Tier 1: Frontier Models										
Grok-4	Text	64	64	64	64	0.10	12366	254.1	–	–
Grok-4	T+V	100	100	100	100	**0.08**	15681	226.0	1.27×	0.89×
GPT-5	Text	80	80	80	80	0.10	11184	187.6	–	–
GPT-5	T+V	98	97	96	96	0.09	12355	284.2	1.10×	1.51×
Tier 2: Prior Strong Baselines										
Claude-Sonnet-4	Text	100	100	100	100	0.58	3747	31.4	–	–
Claude-Sonnet-4	T+V	100	100	100	100	0.77	5299	29.4	1.41×	0.94×
Doubao-Seed	Text	97	97	97	96	**0.25**	8412	121.0	–	–
Doubao-Seed	T+V	98	97	97	98	0.39	8694	104.9	1.03×	0.87×
Tier 3: Efficient/Open Models										
Gemini-2.5-Flash	Text	90	90	90	89	0.77	4369	14.6	–	–
Gemini-2.5-Flash	T+V	89	88	90	89	0.89	7420	12.6	1.70×	0.86×
DeepSeek-V3	Text	96	96	95	96	1.07	3721	66.2	–	–
Kimi-K2	Text	100	100	100	100	1.31	3433	51.9	–	–
Qwen3-VL-4B	Text	12	8	6	4	**0.79**	4917	94.3	–	–
Qwen3-VL-4B	T+V	92	34	23	15	2.81	11522	112.8	2.34×	1.20×
Qwen3-VL-4B (FT)	Text	100	81	91	78	1.15	4321	76.9	–	–
Qwen3-VL-4B (FT)	T+V	100	51	39	35	0.85	10981	91.0	2.54×	1.18×

clock latency can fluctuate due to provider-side batching/routing/caching, we treat **Tok** as the more stable cost axis, while using latency only as a deployment-facing signal rather than a strict monotonic comparator. Under this view, trade-offs are clearly non-monotonic: models with similar VERIFY coverage can differ substantially in verified quality, and models with similar verified quality can require very different token budgets. For example, DeepSeek-V3 and Kimi-K2 achieve near-complete pipeline coverage but with larger gaps, whereas Grok-4 and GPT-5 achieve smaller gaps at much higher cost (Table 2).

Enabling vision makes this tension sharper. Within the same model, $\textbf{VIP}_{\textbf{tok}}$ is consistently > 1 whenever T+V is available, confirming that diagrams reliably increase token usage, while the reliability return is regime-dependent. Frontier models often convert this extra budget into higher end-to-end coverage and/or better gaps, whereas several mid-tier/open models pay a sizable token premium with little or negative VERIFY gain. This motivates reporting VIP alongside stage-wise reliability: it exposes cases where T+V increases cost without improving end-to-end verification.

5.5 Fine-tuning on Qwen

Comparing the base and fine-tuned Qwen rows in Table 2, we find that supervised fine-tuning on intermediate targets substantially improves pipeline survival in the TEXT setting, especially by enforcing the structured output contract and improving downstream executability. Table 3 then focuses on the fine-tuned Qwen3-VL variants under paired TEXT and T+V settings.

For the 4B fine-tuned model, ALIGN remains perfect under both modes, but T+V substantially reduces RECON, FORM, and VERIFY while more than doubling token usage. This suggests that, once entity alignment is controlled, the dominant multimodal bottleneck lies not in binding entities, but in reconstructing the instance and projecting it into solver-executable formulations. Because **Gap** is computed only on the verified subset, the lower gap under T+V should be interpreted together with the drop in VERIFY coverage rather than as an overall improvement.

This split between conditional quality and coverage reinforces why process diagnostics are necessary: improvements at later stages are only useful if upstream reconstruction remains stable under

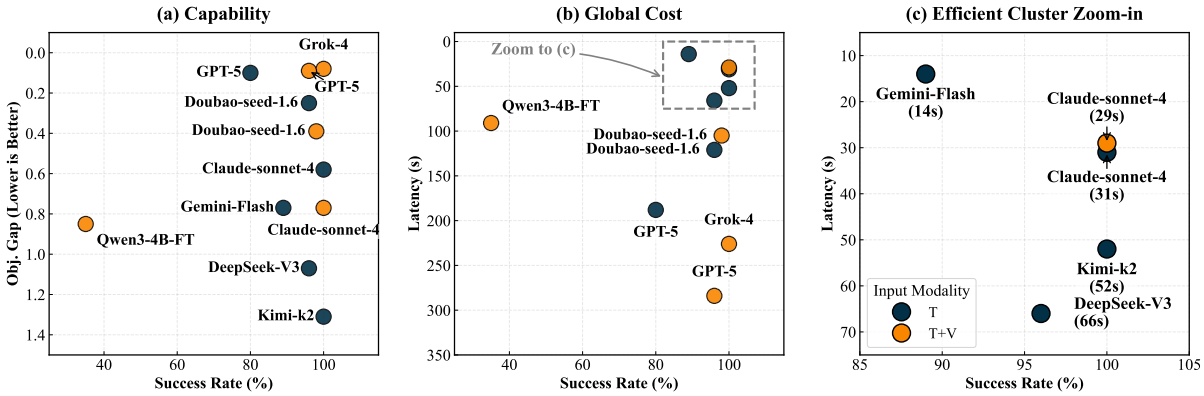


Figure 5: **Reliability–cost trade-offs.** (a) capability view (success vs. gap), (b) global cost view (success vs. latency), and (c) zoom-in on the efficient cluster. Frontier models deliver the best objective quality but incur high latency, while efficient models offer strong success rates at low cost.

Table 3: **Qwen3-VL fine-tuned variants (test, $N = 100$).** Results for fine-tuned Qwen3-VL models under paired TEXT and T+V settings. For the 4B model, vision increases token cost and reduces RECON/FORM/VERIFY, despite a lower conditional **Gap** on the verified subset.

Model	Mode	ALIGN	RECON	FORM	VERIFY	Gap↓	Tok (K)
Qwen3-VL-2B-FT	Text	70	27	5	5	0.88	5.5
Qwen3-VL-2B-FT	T+V	46	25	8	6	0.41	11.6
Qwen3-VL-4B-FT	Text	100	81	91	78	1.15	4.3
Qwen3-VL-4B-FT	T+V	100	51	39	35	0.85	11.0

vision.

5.6 Task-agnostic replication

To test whether our conclusions are tied to max-flow, we replicate the evaluation on **Shortest Path** using the same aligned synthesis pipeline and the same four-layer protocol, changing only the oracle and verification logic (Section 3.2). We observe the same qualitative pattern: end-to-end success is still bottlenecked by intermediate dropout, and vision increases cost while its reliability impact depends on model regime. Full per-model results for shortest path are reported in Appendix D.

5.7 Mechanism analysis of the Visual Paradox

We define a *Visual Paradox* as the regime in which enabling vision both increases inference cost and decreases end-to-end reliability beyond a small tolerance, i.e., $\text{VIP}_{\text{tok}} > 1$ and $S_{\text{VERIFY}}(\text{T+V}) < S_{\text{VERIFY}}(\text{TEXT}) - \delta$. We set $\delta = 2\%$ to avoid treating negligible differences as paradoxical. To understand this behavior, we analyze paired runs of Qwen3-VL-4B-FT under TEXT and T+V on the same instances and under the same strict output contract.

Table 4 isolates truncation-driven contract failures. Under our harness, a run is flagged as cap-hit when generation reaches the configured completion cap, which typically yields incomplete NDJSON and is therefore counted as an output-contract/parsing violation. On Qwen3-VL-4B-FT, T+V exhibits a substantially higher cap-hit rate than TEXT, together with a large increase in mean total tokens. This confirms that truncation explains a substantial part of the end-to-end degradation under multimodal input.

Setting	Non-cap-hit rate	RECON	FORM	VERIFY
TEXT	86%	86.0%	86.0%	80.2%
T+V	44%	93.2%	75.0%	59.1%

Table 4: Stage reliability after excluding completion-cap hits for Qwen3-VL-4B-FT.
Note: "Non-cap-hit rate" denotes the fraction of runs that do not reach the configured completion cap. RECON, FORM, and VERIFY are computed on this non-cap-hit subset only. Thus, the table separates truncation-driven contract failures from the residual stage-level degradation among completed outputs.

However, truncation does not fully explain the Visual Paradox. After excluding cap-hit cases, T+V recovers strongly at RECON, indicating that visual grounding can help conditional instance reconstruction once outputs remain complete. By contrast, T+V still underperforms at FORM and

VERIFY, showing that the remaining degradation is concentrated downstream of reconstruction. In other words, the paradox is best understood as a coupled mechanism: vision increases inference cost and truncation risk, while also making the transfer from visually grounded reconstruction to solver-executable formulation less stable even among non-truncated runs.

This interpretation is also consistent with our paired-case analysis, where the same instance may succeed under TEXT but fail earlier under T+V due to missing reconstruction or non-parsable formulations. We provide representative examples in Table 8 and full metric definitions in Appendix E.

We therefore interpret this analysis as targeted mechanism evidence for a representative paradoxical model under our deployable contract, rather than as a complete causal account for all models, tasks, or rendering conditions.

6 Discussion and Implications

When to enable vision. Diagrams should be a capability-dependent switch, not a default. Frontier models more often gain from visual grounding, improving VERIFY and/or reducing **Gap**, so T+V is reasonable when reliability dominates. For several mid-tier/open checkpoints, vision instead triggers the **Visual Paradox**: VIP_{tok} increases while VERIFY drops, typically because failures move upstream to RECON/FORM. A practical deployment policy is to enable vision only when the model is known to be stable under diagrams, or when the system can fall back to text/structured extraction once validators detect early-stage instability.

Training signals for robust visual modeling. Many multimodal failures come from diagram-to-structure recovery and cross-stage consistency with solver code, not just final solving. This motivates supervision on intermediate artifacts in addition to outcome targets, consistent with our Qwen fine-tuning results. Robustness is also sensitive to the rendering distribution, suggesting value in training/evaluating with rendering perturbations to reduce brittleness.

Generalizing beyond max-flow. GOBench separates view alignment + process diagnostics from task-specific verification. The aligned synthesis and four-layer protocol stay fixed across tasks; extending to Shortest Path replaces only the oracle solver and validators while keeping the same output contract and cost logging. This makes the framework naturally extensible to broader graph-optimization families.

Mechanism and Mitigation. Our stage-wise analysis suggests that multimodal degradation is not monolithic. Under T+V, failures often shift upstream: vision increases input cost and truncation risk, while the remaining non-truncated errors are concentrated in reconstruction-to-formulation transfer rather than final-stage optimization alone. This pattern suggests several practical mitigations. First, systems can use *vision gating* or validator-triggered fallback: if the visual input is likely to inflate context without improving reconstruction, the pipeline can revert to TEXT-only execution or request a simpler visual rendering. Second, because the dominant residual bottlenecks lie in RECON and FORM, intermediate-target supervision is more promising than optimizing only final answers. Third, the sensitivity of non-frontier models to diagram-conditioned reconstruction suggests a need for robustness-oriented training or evaluation under controlled visual perturbations, such as denser layouts, noisier labels, or alternative renderings. More broadly, these findings suggest that multimodal optimization assistants should be deployed conditionally rather than assuming that adding vision is uniformly beneficial.

7 Conclusion

We introduced **GOBench**, an aligned multi-view benchmark for evaluating optimization assistants under realistic multimodal specifications, with solver-derived oracles and a four-layer diagnostic protocol that localizes failures. Across frontier and open-weight models, we found that end-to-end success is frequently bottlenecked by intermediate-stage dropout, and that adding diagrams reliably increases inference cost while its reliability impact is regime-dependent: frontier models often benefit from visual grounding, whereas several mid-tier/open models exhibit a Visual Paradox in which vision degrades executability and verification coverage. By reporting stage-wise reliability together with the **VIP**, GOBench makes reliability–cost trade-offs explicit and exposes cases where multimodality increases cost without improving verified outcomes.

Limitations

GOBench is designed to be task-agnostic in its synthesis and diagnostics, but our empirical coverage is still limited in several ways.

- **Task scope.** Our main-text experiments focus on **Maximum Flow**, and we additionally instantiate the same pipeline on **Shortest Path** by swapping only the oracle solver and task-specific validators while keeping the aligned views and four-layer protocol fixed (Section 3.2). We leave broader NP-hard families to future work, where verification remains solver-grounded but the instance distributions and rendering choices may introduce new failure modes.
- **Visual modality coverage.** Our diagrams are Graphviz-rendered and relatively clean. We expect two findings to be robust to noisier renderings: (i) enabling vision increases input length and thus inference cost (VIP), and (ii) end-to-end success can be bottlenecked by upstream artifact validity under a strict output contract. However, the precise crossover point of the Visual Paradox may shift with rendering noise, which we leave to future work.
- **Serving and prompting effects.** Although we standardize prompts and output contracts across models, results can still be sensitive to provider-side serving variance and instruction details. We therefore treat token-based cost as the most stable proxy and release the evaluation harness for reproducibility.

Ethics Statement

GOBench uses procedurally generated synthetic instances, so it contains no personally identifiable information and does not reproduce proprietary real-world documents. We report inference cost because multimodal evaluation and deployment incur non-trivial compute and energy overhead; releasing the dataset and harness aims to reduce redundant benchmarking by enabling direct, reproducible comparisons. Finally, while optimization assistants can be applied in many domains, our benchmark tasks are generic graph formulations and do not target sensitive operational settings; we nonetheless encourage practitioners to follow domain-appropriate safety policies when deploying model-generated optimization pipelines. To facilitate reproducibility and future research, the GOBench dataset, generation scripts, and evaluation harness are released under the **Apache License 2.0**. We confirm that the artifacts were generated using open-source tools (NetworkX [BSD], Graphviz [EPL]) consistent with their intended research use. The derived GOBench dataset is released under Apache-2.0, which is fully compatible with using these upstream tools for data generation.

We employed Large Language Models as the primary subjects of our experiments, as detailed in Section 5, Table 2. Additionally, we used general-purpose AI assistants (ChatGPT, Gemini) solely for minor language polishing and checking LaTeX syntax. All scientific claims and ideas are our own.

Acknowledgments

The research is partially supported by Quantum Science and Technology-National Science and Technology Major Project (QNMP) 2021ZD0302900, China National Natural Science Foundation with No. 92567301, 62132018, 62231015, "Pioneer" and "Leading Goose" R&D Program of Zhejiang, 2023C01029 and 2023C01143, and Anhui Provincial Science and Technology Innovation Program 202523o09050019.

References

Lingjiao Chen, Matei Zaharia, and James Zou. 2024. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*. Featured Certification.

Antonia Creswell and Murray Shanahan. 2022. Faithful reasoning using large language models. *arXiv preprint arXiv:2208.14271*.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. 2023. Faith and fate: Limits of transformers on compositionality. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Chaoyou Fu, Peixian Chen, Yunhang Shen, Yulei Qin, Mengdan Zhang, Xu Lin, Jinrui Yang, Xiawu Zheng, Ke Li, Xing Sun, Yunsheng Wu, Rongrong Ji, Caifeng Shan, and Ran He. 2025. MME: A comprehensive evaluation benchmark for multimodal large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Emden R Gansner and Stephen C North. 2000. An open graph visualization system and its applications to soft-

ware engineering. *Software: practice and experience*, 30(11):1203–1233.

Aric Hagberg, Pieter J Swart, and Daniel A Schult. 2007. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Laboratory (LANL).

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

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

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. 2025b. Llms for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 2678–2710.

Beibin Li, Konstantina Mellou, Bo Zhang, Jeevan Pathuri, and Ishai Menache. 2023. Large language models for supply chain optimization. *Preprint*, arXiv:2307.03875.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D Manning, Christopher Re, Diana Acosta-Navas, Drew A. Hudson, and 31 others. 2023. Holistic evaluation of language models. *Transactions on Machine Learning Research*. Featured Certification, Expert Certification, Outstanding Certification.

Fangyu Liu, Julian Eisenschlos, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Wenhu Chen, Nigel Collier, and Yasemin Altun. 2023. Deplot: One-shot visual language reasoning by plot-to-table translation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 10381–10399.

Minesh Mathew, Dimosthenis Karatzas, and CV Jawahar. 2021. Docvqa: A dataset for vqa on document images. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 2200–2209.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nova, and 1 others. 2021. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212.

Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–14.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and 1 others. 2023. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 competition track*, pages 189–203. PMLR.

Shang-Hua Teng. 1999. Coarsening, sampling, and smoothing: Elements of the multilevel method. In Michael T. Heath, Abhiram Ranade, and Robert S. Schreiber, editors, *Algorithms for Parallel Processing*, pages 247–276. Springer New York, New York, NY.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. 2023. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. 2023. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *Advances in Neural Information Processing Systems*, 36:23826–23854.

Yan Zeng, Hanbo Zhang, Jiani Zheng, Jiangnan Xia, Guoqiang Wei, Yang Wei, Yuchen Zhang, Tao Kong, and Ruihua Song. 2024. What matters in training a gpt4-style language model with multimodal inputs? In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7930–7957.

Zhuosheng Zhang, Aston Zhang, Mu Li, hai zhao, George Karypis, and Alex Smola. 2024. Multimodal chain-of-thought reasoning in language models. *Transactions on Machine Learning Research*.

A Visual Inference Penalty (VIP) and Infrastructure Details

In the main text, we defined the Visual Inference Penalty (VIP) as the multiplicative overhead incurred by enabling visual modalities. This appendix details the exact calculation methodology and the experimental infrastructure used for open-weight models.

A.1 Token Counting Methodology

Unlike heuristic estimation methods, we utilize **server-side reporting** to ensure precise measurements of the "Visual Token" cost.

Data Acquisition. For both proprietary APIs and local vLLM deployments, we extract token counts directly from the standard OpenAI-compatible response object. specifically the `usage` field:

$$T_{\text{total}} = T_{\text{prompt}} + T_{\text{compl}} \tag{11}$$

where T_{prompt} and T_{compl} correspond to the `prompt_tokens` and `completion_tokens` fields in the API response, respectively. This approach accounts for the exact number of visual tokens (image embeddings) projected by the model's vision encoder (e.g., ViT) into the LLM's context window.

Image Preprocessing. To standardize evaluation across models with varying aspect ratio constraints (e.g., Doubao models requiring ratios between 1:3 and 3:1), all visual inputs in GOBench are preprocessed to a **1:1 aspect ratio**. We rely on the model's native processor (e.g., `AutoProcessor` in HuggingFace/vLLM) to handle resolution scaling, ensuring the benchmark reflects default usage behaviors.

A.2 Experimental Infrastructure

All open-weight model evaluations and fine-tuning were conducted on a local high-performance computing node.

Hardware Configuration.

- **GPU:** 4 $\times$ NVIDIA GeForce RTX 4090 (24GB VRAM per GPU).
- **Interconnect:** PCIe Gen 4.0.

Inference Backend (vLLM). We deploy models using the `vLLM` engine to simulate production-grade throughput. Key deployment parameters include:

- **Tensor Parallelism (TP):** Set to $TP = 4$ to distribute large models across all available GPUs.
- **Memory Optimization:** We enable `-kv-cache-dtype fp8` to maximize the KV cache capacity.
- **Context Window:** The maximum model length is clamped to `16,384` tokens.

Fine-Tuning Setup. For the FT experiments (e.g., Qwen3-VL-FT), we utilize the LLaMA-Factory framework. Training is performed using LoRA (Hu et al., 2022) with 4-bit quantization (`-quantization_bit 4`) and Flash Attention 2 (`fa2`) to fit within the consumer-grade VRAM constraints.

A.3 Latency Measurement

Latency is recorded as the end-to-end wall-clock time from the initial HTTP request to the receipt of the final token, averaged over the test set. This captures the full cost of visual encoding, transmission, and autoregressive generation.

Log Analysis: As shown in Listing A.1, the deployment utilizes `tensor_parallel_size=4` and `kv_cache_dtype='fp8'` to optimize throughput on the 4$\times$4090 cluster. The *Prefix cache hit rate* stabilizes at $\approx$36%, indicating effective reuse of the system prompt and few-shot examples across requests, verifying the efficiency claims in Section 4.

B Data Generation Pipeline

This appendix details the procedural pipeline used to generate the aligned structural, textual, and visual views for GOBench.

B.1 Structural Synthesis

To ensure that every instance represents a non-trivial optimization problem (i.e., Max Flow > 0), we eschew standard random graph generators (e.g., Erdos-Renyi) in favor of a custom **Path-First Construction** algorithm implemented in Python.

Generation Logic. The algorithm proceeds in two phases:

1. **Forced Reachability:** We first sample a random simple path from source s to sink t. This mathematically guarantees feasibility.

Listing A.1: vLLM Server Deployment Log (Qwen3-VL-4B-FT)

```
[INFO 12-20 20:27:54] vLLM API server version 0.12.0
[INFO 12-20 20:27:54] args: {
'model': '.../saves/qwen3-vl_4b-instruct-finetune',
'tensor_parallel_size': 4, # 4-GPU Parallelism
'gpu_memory_utilization': 0.6,
'kv_cache_dtype': 'fp8', # FP8 Optimization Enabled
'max_model_len': 16384
}
[INFO 12-20 20:27:54] Using fp8 data type to store kv cache.
[INFO 12-20 20:27:54] Chunked prefill is enabled.
...
[INFO 12-20 20:28:13] Starting to load model...
[INFO 12-20 20:28:13] Using FLASHINFER attention backend.
[INFO 12-20 20:28:14] Loading safetensors checkpoint shards: 100% Completed
...
[INFO 12-20 20:32:09] Engine 000: Avg gen throughput: 12.7 tokens/s, Prefix cache hit: 0.0%
[INFO 12-20 20:32:59] Engine 000: Avg gen throughput: 34.4 tokens/s, Prefix cache hit: 18.4%
... (warmup phase) ...
[INFO 12-20 21:16:29] Engine 000: Avg gen throughput: 34.8 tokens/s, Prefix cache hit: 35.5%
[INFO 12-20 21:17:19] Engine 000: Avg gen throughput: 35.7 tokens/s, Prefix cache hit: 35.5%
[INFO 12-20 21:19:29] Engine 000: Avg gen throughput: 35.0 tokens/s, Prefix cache hit: 35.4%
...
[INFO 12-20 22:40:12] Application shutdown complete.
```

Log Analysis: As shown in Listing A.1, the deployment utilizes `tensor_parallel_size=4` and `kv_cache_dtype='fp8'` to optimize throughput on the 4×4090 cluster. The *Prefix cache hit rate* stabilizes at ≈36%, indicating effective reuse of the system prompt and few-shot examples across requests.

2. **Random Edge Densification:** We subsequently sample node pairs (u, v) to add directed edges until a target density is met, strictly enforcing no self-loops or multi-edges.

B.2 Visual Rendering Settings

Visual inputs are rendered using **Graphviz** with the `dot` layout engine. As shown in Listing B.2, we enable `splines='curved'` to reduce visual clutter and use a deterministic color hashing strategy for edges to test the model's ability to trace specific connections.

B.3 Dataset Distribution

Figure 7 illustrates the distribution of key topological properties across the 1,000 generated instances.

B.4 Solver Oracle Implementation

To verify model outputs, we implement a computational oracle. We compile each graph instance $G = (V, E, C)$ into a canonical AMPL model using the `amplpy` interface and solve it with **Gurobi** to obtain the ground-truth objective z^*.

B.5 Data Artifact Samples

The pipeline transforms a raw topological seed into aligned multimodal views and a solver-compatible format. Figure 8 (next page) displays the generated text and image, while Figure 9 shows the underlying data structures.

C Fine-Tuning Implementation Details

To bridge the capability gap in open-weight models, we performed Supervised Fine-Tuning (SFT) on the Qwen-VL and LLaVA families. This section documents the training configuration, dynamics, and infrastructure to ensure reproducibility.

C.1 Training Framework

We utilized the `LLaMA-Factory` framework for unified efficient fine-tuning. The training pipeline was optimized for consumer-grade hardware (4×RTX 4090) using the following techniques:

- **QLoRA:** 4-bit quantization (NF4) for the base model to reduce VRAM usage.
- **Flash Attention 2:** Accelerated attention computation for Qwen series models.
- **Gradient Checkpointing:** Enabled to fit longer context lengths $(16,384)$.

C.2 Hyperparameter Configuration

We employed Low-Rank Adaptation (LoRA) targeting all linear modules in the attention blocks.

Listing B.1: Graph Generation Hyperparameters (Python)

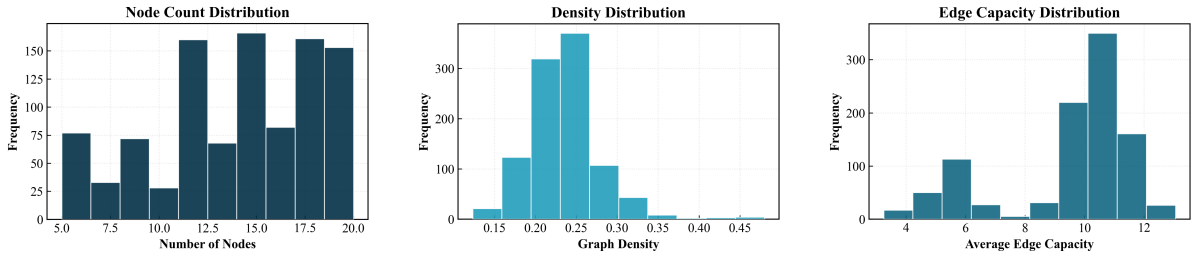
```
# Configuration for "Path-First" Construction
GEN_CONFIG = {
    "n_nodes_min": 5,       "n_nodes_max": 20,
    "m_edges_min": 6,       "m_edges_max": 113,
    "cap_min": 1,           "cap_max": 20,
    "allow_self_loops": False,
    "allow_multi_edges": False,
    "force_st_connectivity": True   # Crucial for valid max-flow
}
```

Listing B.2: Graphviz Rendering Configuration

```
# Initializing the Digraph with 'dot' engine
g = Digraph('G', engine='dot')
g.attr(rankdir='TB', splines='curved', overlap='false', nodesep='0.6', ranksep='1.0')

# Dynamic Edge Coloring based on categorical hash
# We map edge attributes to a categorical palette to test visual grounding
color = PALETTE[(u * 31 + v * 17 + cap * 7) % len(PALETTE)]
g.edge(str(u), str(v), color=color, fontcolor=color, penwidth='2')
```

Figure 6: Configuration details for the structural generation (top) and visual rendering (bottom) pipelines.



(a) Node Count (b) Graph Density (c) Avg Edge Capacity

Figure 7: **Dataset Distribution.** Histograms showing the diversity of graph scale and density.

The visual encoders were frozen to preserve pre-trained alignment features. Table 5 details the specific hyperparameters used across all experiments.

C.3 Training Dynamics

Figure 10 visualizes the training loss trajectories extracted from the experimental logs.

- **Convergence:** All Qwen-series models (Blue lines) demonstrated rapid convergence, reaching a loss < 0.15 within 6 epochs.
- **Stability:** The **Qwen3-VL-2B** (Deep Blue) showed exceptional stability despite its smaller parameter size, achieving the lowest final loss alongside Qwen2-VL-7B.
- **Comparison:** LLaVA-1.5-7B (Orange Red) converged slower and settled at a higher loss (≈ 0.17), correlating with its lower performance in the RECON layer.

D Shortest Path Results

We replicate the full GOBench evaluation on **Shortest Path** using the same aligned synthesis pipeline and four-layer protocol (ALIGN/RECON/FORM/VERIFY), swapping only the task oracle and verification logic (Section 3.2). Metrics follow Section 4.4. Table 6 reports results on the held-out test split ($N = 100$). Models that do not support vision are reported with TEXT-only rows and have no VIP entries.

End-to-end success Bottleneck (P2). Shortest Path exhibits the same process bottleneck as max-flow: end-to-end success depends on whether intermediate artifacts survive to the solver check. Fron-

Intermediate Artifacts: Aligned Multimodal Views (t, v)

[Text Description (t)]
Context: This is a national logistics network connecting Beijing, Tokyo, Dakar, Queensland, Jakarta, Quito, Helsinki, Manila, Houston and Prague.
Task: Find max flow from Beijing to Manila.
Edges:
- Route Beijing → Manila (cap 6).
- Route Beijing → Houston (cap 3).
- Route Beijing → Prague (cap 3).
- Route Tokyo → Quito (cap 1).
- Route Dakar → Tokyo (cap 8).
- ... (16 more edges omitted for brevity)

[Visual Diagram (v)]

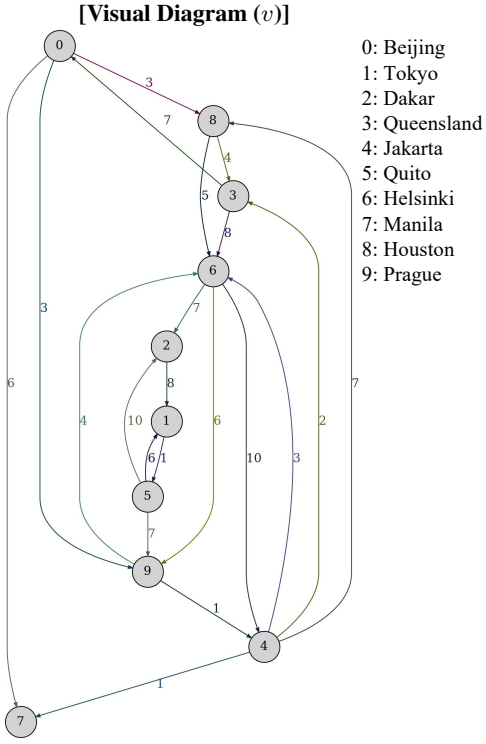


Figure 8: **Generated Multimodal Views.** The text description (left) and Graphviz-rendered diagram (right) are generated from the same raw seed, ensuring strict alignment.

Table 5: **Hyperparameters for SFT.** Settings derived from the `run_all.sh` execution log.

Hyperparameter	Value
LoRA Configuration	
Rank (r)	64
Alpha (α)	128
Dropout	0.05
Target Modules	`all-linear`
Optimization	
Learning Rate	1e-5
Scheduler	Cosine
Optimizer	AdamW (8-bit)
Batch Size	1 (per device)
Gradient Accumulation	8
Effective Batch Size	**32** (4 GPUs × 1 × 8)
Epochs	15
Max Length	16,384

tier models sustain near-complete survival across all four stages (e.g., Grok-4 remains 99% at every layer under both modes), whereas efficient models show substantial attrition before verification. In particular, Gemini-Flash drops from 77% at ALIGN/RECON/FORM to 64% at VERIFY even in text-only mode, and collapses further under T+V (44% at early stages, 36% at VERIFY). These gaps would be blurred by outcome-only reporting; stage-wise reliability isolates whether failures arise from structure recovery and formulation rather than from the solver check itself.

Trade-offs and VIP (P3). Shortest Path also reproduces the non-monotonic deployment frontier: higher token/latency budgets do not guarantee higher VERIFY success or lower **Gap**. Within the same model, enabling vision always increases token cost , but the reliability/quality return varies widely. For Grok-4, T+V yields a modest gap improvement at a small token premium with unchanged verification coverage. For GPT-5, the token premium is mild but latency doubles while both **Gap** and coverage slightly degrade. For Claude-Sonnet-4, vision increases tokens substantially and reduces verification coverage even though conditional gap slightly improves. This supports the main-text recommendation that token-based VIP is the most stable proxy for multimodal overhead, and that cost must be reported together with VERIFY coverage and **Gap**.

Listing B.3: Raw Structural Seed (File: graph11.txt)

```
10 21          # Metadata: N=10 nodes, M=21 edges
0 7 6          # Edge: 0 -> 7, Capacity=6
0 8 3          # Edge: 0 -> 8, Capacity=3
0 9 3          # Edge: 0 -> 9, Capacity=3
1 5 1          # ... (list continues) ...
9 4 1          # Last Edge
0 7            # Task Definition: Source=0, Sink=7
```

Listing B.4: Compiled AMPL Data Block (Oracle Input)

```
data;
param n := 10;
param source := 0;
param sink := 7;

set NODES := 0 1 2 3 4 5 6 7 8 9;

set EDGES :=
   (0,7) (0,8) (0,9) (1,5) (2,1) (3,0) (3,6) (4,7) (4,8) (4,6)
   (4,3) (5,2) (5,9) (5,1) (6,2) (6,9) (6,4) (8,6) (8,3) (9,6) (9,4);

param capacity :=
   0 7 6   0 8 3   0 9 3   1 5 1   2 1 8   3 0 7   3 6 8   4 7 1
   4 8 7   4 6 3   4 3 2   5 2 10  5 9 7   5 1 6   6 2 7   6 9 6
   6 4 10  8 6 5   8 3 4   9 6 4   9 4 1;
```

Figure 9: **Data Artifacts.** The raw topological seed (top) is the ground truth. It is compiled into the AMPL data format (bottom) for the solver oracle.

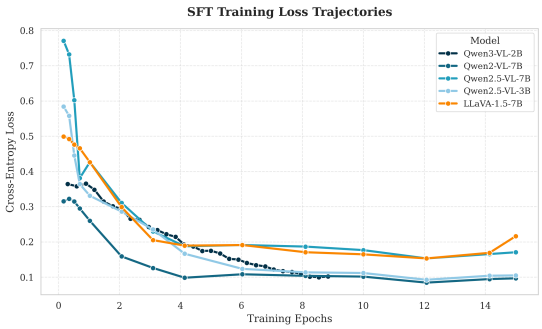


Figure 10: **SFT Training Loss Curves.** Comparison of convergence rates across 5 open-weight models. The Qwen-VL family (Blue shades) consistently outperforms LLaVA (Orange Red) in fitting the graph optimization data distribution.

Modality effects reverse under vision (P1). Comparing TEXT vs. T+V isolates the causal effect of adding the aligned diagram view. On Shortest Path, the modality effect remains regime-dependent: frontier models can incorporate diagrams without destabilizing intermediate artifacts, while efficient models exhibit a clear Visual Paradox where diagrams increase cost but reduce reliability. Gemini-Flash is the clearest example: T+V more than doubles tokens (VIP_{tok}=2.37×) while reducing early-stage survival from 77% to 44%, which propagates to a drop in VERIFY coverage (64% → 36%) and worse verified gap (0.634→0.872). This indicates that, for weaker regimes, diagrams can act as distractions that destabilize reconstruction/formulation rather than providing usable grounding.

Differences between max-flow and shortest path. While the qualitative conclusions are consistent across tasks, the VERIFY stage is *mechanistically different*, which changes the dominant failure signatures. In **Max-Flow**, verification is constraint-heavy: feasibility is governed by global flow conservation and capacity constraints over many edges, so a single reconstruction error can cascade into many violations (non-existent edges, capacity overflow, and conservation breaks). In **Shortest Path**, verification is structure- and objective-consistency–heavy: a candidate can be a syntactically valid path yet fail because it is not a coherent walk or because the model's stated objective disagrees with the cost implied by its own path, which inflates **Gap** even when a plausible path is

Table 6: **Shortest Path results on GOBench (test, $N = 100$).** Stage-wise reliability reports the fraction of instances producing valid artifacts: ALIGN, RECON, FORM, VERIFY. **Gap↓** is the mean *absolute* normalized objective gap on the verified subset. **Tok** denotes mean total tokens. **VIP** reports the multiplicative cost of enabling vision, measured in tokens and latency; values < 1 may occur due to serving variance.

Model	Mode	ALIGN	RECON	FORM	VERIFY	Gap↓	Tok	Lat(s)	VIP_{tok}	VIP_{lat}
Tier 1: Frontier Models										
Grok-4	Text	99	99	99	99	0.033	2890	27.7	–	–
Grok-4	T+V	99	99	99	99	0.027	3129	27.7	1.08×	1.00×
GPT-5	Text	98	98	98	98	0.054	2846	21.3	–	–
GPT-5	T+V	97	97	97	97	0.080	3033	42.5	1.07×	2.00×
Tier 2: Prior Strong Baselines										
Claude-Sonnet-4	Text	100	100	100	99	0.167	2195	10.0	–	–
Claude-Sonnet-4	T+V	100	100	100	96	0.159	3690	13.7	1.68×	1.37×
Tier 3: Efficient/Open Models										
Gemini-2.5-Flash	Text	77	77	77	64	0.634	1249	6.1	–	–
Gemini-2.5-Flash	T+V	44	44	44	36	0.872	2961	7.4	2.37×	1.22×

produced. Cost behavior also differs in practice: because Shortest Path often elicits longer natural-language rationales or path descriptions under the same output contract, some providers show larger T+V token/latency inflation, reinforcing that vision overhead depends jointly on serving behavior and task-specific token footprints. Despite these task-specific mechanisms, the core findings remain invariant: end-to-end success is bottlenecked by intermediate dropout, and the reliability–cost return of vision is regime-dependent.

E Per-instance log schema and metrics

In all analyses, we follow an earliest-invalid-stage convention: downstream indicators are only evaluated when all upstream artifacts are valid, so errors caused by upstream failures are not double-counted. Each evaluation instance is stored as one JSON record. We summarize the fields used in our analysis.

Identifiers. `index` is the instance index in the split; `split` indicates Train/Dev/Test; `model` is the checkpoint/API name; `vlm` $\in \{0, 1\}$ indicates whether the aligned diagram is provided.

Identifiers.

1. `index`: instance index within the split file.
2. `split`: dataset split identifier (`train`/`dev`/`test`).
3. `model`: checkpoint or API endpoint name.
4. `vlm` $\in \{0, 1\}$: whether the aligned diagram view is provided (1 = T+V, 0 = TEXT).

Runtime and cost.

1. `perf.latency_ms_total`: end-to-end wall-clock latency (ms).
2. `perf.prompt_tokens`, `perf.completion_tokens`, `perf.total_tokens`: provider-reported token counts for the prompt, completion, and their sum.
3. We use token-based VIP as the primary cost proxy; latency-based VIP can be noisier due to serving-side variance (batching, routing, caching).

Top-level summaries (`summary`).

1. `summary.graph_score` and `summary.ampl_sim`: compact summaries of reconstruction and formulation quality (see metric definitions below).
2. `summary.objective_gap`: signed objective gap reported by the harness; in all aggregate reports we use |`objective_gap`|.
3. `summary.feasible` (when present): feasibility of the predicted solution under solver-grounded verification (used as VERIFY success).

Artifact presence flags.

1. `has_MAP`, `has_GRAPH`, `has_AMPL`, `has_SOL`: whether the corresponding artifact was emitted and parsed into the expected schema.
2. `parse_violations`: output-contract violations detected by the harness.

Graph reconstruction metrics (`metrics.GRAPH`). Let $E^\star$ be the oracle edge set and $\hat{E}$ be the predicted edge set after applying the anchored mapping. We report:

(i) `edges_precision`, `edges_recall`, `edges_F1`: edge recovery quality comparing $\hat{E}$ against $E^\star$.

(ii) `cap_MAE` / `w_MAE` and `cap_MAPE` / `w_MAPE`: absolute/relative parameter error on matched edges.

(iii) `source_acc` and `sink_acc`: correctness of the query endpoints.

(iv) `nodes_index_ok`: whether node indices are consistent with the anchored mapping.

(v) `N_match`: a boolean summary of mapping/index consistency used by the validator.

(vi) `graph_score`: an aggregate reconstruction score, also stored as `summary.graph_score`.

Formulation metrics (`metrics.AMPL`). We report compilation and data/structure consistency between the produced AMPL program and the oracle instance:

(i) `parsed_ok`: whether the AMPL program compiles under the harness.

(ii) `structure_score`: whether the formulation instantiates the correct variable/constraint template.

(iii) `edges_jaccard`: overlap between edges implied by the AMPL data section and the oracle edge set.

(iv) `capacity_MAPE` / `weight_MAPE`: parameter mismatch in the AMPL data section.

(v) `data_score`: a summary score of AMPL data fidelity.

(vi) `ampl_sim`: an aggregate similarity score, also stored as `summary.ampl_sim`.

(vii) (When available) `dat_obj`: objective implied by AMPL data; `ampl_gap`: signed mismatch between AMPL-implied objective and the oracle.

Solver/solution metrics (`metrics.SOL`). We report feasibility, objective quality, and constraint violations for the predicted solution:

(i) `feasible`: the indicator used for VERIFY success in this paper.

(ii) `pred_obj` and `oracle_obj`: objectives computed from the predicted solution and the oracle.

(iii) `objective_gap`: signed normalized objective gap; we use $|\cdot|$ for aggregates.

(iv) `violations`: a list of violated constraints.

(v) Task-specific fields may appear, e.g., `flow_edge_F1` for max-flow edge support or `path_match` for shortest-path.

Stage-wise indicators used in tables. Let i index instances and let (m, μ) denote model and mode. We define:

(i) $a_i = \mathbb{I}[\texttt{has\_MAP}]$,

(ii) $r_i = \mathbb{I}[\texttt{has\_GRAPH}]$,

(iii) $f_i = \mathbb{I}[\texttt{has\_AMPL} \wedge \texttt{metrics.AMPL.parsed\_ok}]$,

(iv) $v_i = \mathbb{I}[\texttt{metrics.SOL.feasible}]$.

Objective gap is aggregated on the verified subset ($v_i = 1$), using absolute values. Here is an example figure 11.

E.1 Uncertainty of Conditional Metrics on MaxFlow

Table 7 reports uncertainty estimates for **conditional** metrics on MaxFlow (test). Following the main table, VERIFY coverage is counted by `has_SOL=true`, i.e., whether the pipeline produced a solver-checkable solution artifact. We then summarize the **Gap** distribution on the same verified subset using (i) the mean and sample variance, and (ii) an interval estimate: a 95% bootstrap CI for the mean when $n_{\text{VERIFY}} \geq 5$, otherwise the empirical range. We additionally report the sample variance of per-instance latency, which reflects backend variance in serving time.

Coverage governs interpretability. The verified subset size varies substantially across models, and thus directly affects how stable conditional estimates are. Frontier and strong API models provide large verified subsets, Grok-4 T+V and Claude-Sonnet-4 both reach $n_{\text{VERIFY}} = 100$, GPT-5 reaches $n_{\text{VERIFY}} = 80$ in Text and 96 in T+V, yielding comparatively tight mean-gap intervals. In contrast, Qwen3-VL-4B (Text) has only $n_{\text{VERIFY}} = 4$, so its Gap interval expands to a wide range, indicating that conditional quality estimates are not statistically stable at such low coverage.

Gap uncertainty aligns with reliability regimes. For models with high verification coverage, the Gap intervals are narrow relative to their mean values, suggesting that conditional quality is well-estimated. For instance, Grok-4 shows low Gap means with tight intervals, and GPT-5 exhibits similarly small Gap values under both modes. Conversely, models with higher Gap variance show substantially wider intervals, consistent with more heterogeneous instance-level behavior even when coverage is high.

Vision can trade coverage for conditional quality. Comparing paired Text vs. T+V rows shows that enabling vision can change both coverage and the conditional Gap distribution. For example, Qwen3-VL-4B exhibits a pronounced coverage increase from Text to T+V, but also a large increase in mean Gap and variance, implying that the subset that becomes "verifiable" under vision remains noisy in objective quality. For Qwen3-VL-4B (FT), T+V reduces conditional Gap mean relative to Text, while coverage drops, illustrating why the main paper interprets Gap jointly with VERIFY: lower conditional Gap does not imply better end-to-end performance if fewer instances reach verification.

Latency variance is substantial and model-dependent. Latency variance spans multiple orders of magnitude, particularly for frontier models, reflecting provider-side batching/routing variability and heterogeneous instance difficulty. This motivates our emphasis on token-based cost as the more stable proxy in the main analysis, while using latency primarily as a deployment-facing signal.

F Prompts and Instruction Templates

To ensure rigorous evaluation and enforce the structured output format required by our 4-layer diagnostic pipeline, we employed a unified prompt template across all models (both LLMs and VLMs). 12

F.1 System Instruction

We utilize a lightweight system prompt to avoid over-constraining the model's inherent reasoning capabilities while establishing a helpful persona.

F.2 User Instruction Template

The user prompt is designed to serve two purposes: (1) defining the strictly formatted NDJSON output schema, and (2) providing the specific problem instance data. For Visual Language Models (VLMs), the image token is prepended to the text instruction.

G Evaluated Models and Representative Outputs

In this section, we first list all the models evaluated in GOBench (Section G.1) and then present representative raw outputs to illustrate the diagnostic capability of our pipeline (Section G.2).

G.1 Evaluated Model Catalog

We evaluated a comprehensive set of state-of-the-art models, ranging from proprietary APIs to open-weight models fine-tuned in this work. Table 9 categorizes these models by family and capability.

G.2 Representative Output Analysis

To illustrate the diagnostic depth of the GOBench pipeline, we present raw outputs from two representative models on the same test instance.

H Metric Implementation Details

This appendix provides the mathematical definitions and implementation logic for the metrics used in the GOBench evaluation pipeline. The evaluation is automated, deterministic, and utilizes a standard solver oracle to verify semantic correctness.

H.1 Layer 2: Graph Reconstruction

This layer evaluates the model's ability to ground natural language into a structured topology.

Structural Similarity (Edge F1). Since the node mapping is explicitly defined in Layer 1, we evaluate structural accuracy using the F1-score of the edge set. Let E_{pred} be the set of directed edges (u, v) predicted by the model and E_{ref} be the ground truth edges.

Table 7: Uncertainty for conditional metrics on MaxFlow (test). We report VERIFY coverage (VERIFY%, n_{VERIFY}; counted by `has_SOL=true`), the mean absolute objective gap on verified instances (Gap = |`objective_gap`| computed on the same `has_SOL` subset) and its sample variance, and a Gap interval: 95% bootstrap CI for the mean when $n_{\text{VERIFY}} \geq 5$, otherwise the range over verified instances; '–' indicates no verified instances. We also report the sample variance of per-instance latency (`latency_ms_total`).

Model	Setting	VERIFY%	n_{VERIFY}	Gap mean	Gap var	Gap interval	Latency var
Claude-Sonnet-4	TEXT	100.0	100	0.5815	1.35e+00	[0.3972, 0.8143]	4.49e+08
Claude-Sonnet-4	T+V	100.0	100	0.7666	4.28e+00	[0.4476, 1.2240]	5.73e+07
DeepSeek-V3	TEXT	96.0	96	1.0675	8.34e+00	[0.5726, 1.6861]	7.05e+08
Doubao-Seed	TEXT	96.0	96	0.2526	1.08e-01	[0.1953, 0.3172]	1.77e+09
Doubao-Seed	T+V	98.0	98	0.3910	2.75e-01	[0.2863, 0.5015]	3.93e+08
GPT-5	TEXT	80.0	80	0.0961	6.34e-02	[0.0482, 0.1535]	1.22e+10
GPT-5	T+V	96.0	96	0.0876	5.44e-02	[0.0437, 0.1349]	1.33e+10
Gemini-2.5-Flash	TEXT	89.0	89	0.7743	1.85e+00	[0.5354, 1.0780]	2.63e+08
Gemini-2.5-Flash	T+V	89.0	89	0.8917	4.05e+00	[0.5532, 1.3799]	1.28e+07
Grok-4	TEXT	64.0	64	0.1012	7.13e-02	[0.0420, 0.1693]	2.66e+10
Grok-4	T+V	100.0	100	0.0769	5.25e-02	[0.0334, 0.1237]	4.92e+09
Kimi-K2	TEXT	100.0	100	1.3107	1.34e+01	[0.6765, 2.1024]	5.50e+08
Qwen3-VL-4B	TEXT	4.0	4	0.7877	5.81e-01	[0.0769, 1.7907]	9.58e+08
Qwen3-VL-4B	T+V	15.0	15	2.8064	1.77e+01	[0.9984, 5.0505]	7.11e+08
Qwen3-VL-4B (FT)	TEXT	78.0	78	1.1470	6.68e+00	[0.6625, 1.8039]	7.03e+08
Qwen3-VL-4B (FT)	T+V	35.0	35	0.8467	1.94e+00	[0.4731, 1.3313]	1.04e+09

Table 8: **Paired cases (Qwen3-VL-4B-FT) illustrating the Visual Paradox.** Cases are matched by `index` across TEXT (`vlm=false`) and T+V (`vlm=true`). We report the earliest failing stage (Appendix E), total tokens, latency, and whether generation hits the completion cap (4096).

Index	Text Earliest	T+V Earliest	Tok (Text)	Tok (T+V)	Lat (Text)	Lat (T+V)	Notes
12	OK	Fail@RECON	4074	11628	70.8s	115.0s	T+V hits completion cap `has_GRAPH` missing.
80	OK	Fail@FORM	4207	10436	69.6s	54.6s	T+V has GRAPH `metrics.AMPL.parsed_ok=false`.
1	Fail@VERIFY	Fail@RECON	6141	12104	115.1s	115.6s	Text reaches solver check but infeasible T+V truncates and fails earlier.

$$\text{MAPE} = \frac{1}{|I|} \sum_{(u,v)\in I} \frac{|C_{pred}(u,v) - C_{ref}(u,v)|}{\max(1, C_{ref}(u,v))} \tag{12}$$

Composite Graph Score. To measure overall reconstruction quality, we implement a weighted scoring function (as defined in `eval_graph`):

$$S_{graph} = 0.45 \cdot F1 + 0.30 \cdot (1 - \text{MAPE}_{clip}) + 0.15 \cdot \mathbb{I}_{idx} + 0.10 \cdot \text{Acc}_{S/T} \tag{13}$$

where MAPE_{clip} is clamped to $[0, 1]$, $\mathbb{I}_{idx}$ is a binary indicator for valid contiguous node indexing, and $\text{Acc}_{S/T}$ measures source/sink identification accuracy.

H.2 Layer 3: Formal Modeling (AMPL Gap)

To verify the semantic correctness of the generated optimization code, we define the *AMPL Gap*. We extract the parameter sets (N, E, C, s, t) from the generated `dat` block, inject them into a canonical Max-Flow AMPL model, and solve it to obtain the objective value z_{ampl}.

$$\text{Gap}_{ampl} = \frac{z_{ampl} - z^*}{\max(1, |z^*|)} \tag{14}$$

where z^* is the ground truth optimal value provided by the dataset oracle.

H.3 Layer 4: Solution Verification

The final layer evaluates the validity of the direct numerical solution. A solution is marked as **Fea-**

Listing E.2: Diagnostic Log for Claude-Sonnet-4 (ShortestPath, 1-shot)


```
{
  "id": "shortest_path-hard-graph228",
  "model": "anthropic/claude-sonnet-4",
  "vlm": false,
  "perf": { "latency_ms_total": 10948, "prompt_tokens": 865, "completion_tokens": 716, "total_tokens": 1581 },
  "summary": { "graph_score": 1.0, "ampl_sim": 1.0, "feasible": true, "objective_gap": 0.0 },
  "metrics": {
    "GRAPH": { "edges_F1": 1.0, "graph_score": 1.0 },
    "AMPL": { "parsed_ok": true, "structure_score": 1.0, "ampl_sim": 1.0, "ampl_gap": 0.0 },
    "SOL": {
      "feasible": true,
      "pred_obj": 27.0,
      "oracle_obj": 27.0,
      "objective_gap": 0.0,
      "path_match": 1
    }
  }
}
```


Figure 11: A 1-shot diagnostic log for Claude-Sonnet-4 on a ShortestPath instance. Although the reconstructed graph and solver formulation are perfect and the predicted path matches the oracle (`objective_gap=0`), the pipeline flags a self-consistency violation: the model *states* an incorrect objective value that disagrees with the computed path cost.

sible if and only if it satisfies three physical constraints (with tolerance $\epsilon = 10^{-6}$):

1. **Capacity Constraint:** $\forall (u, v), 0 \leq f_{uv} \leq c_{uv}$.
2. **Flow Conservation:** $\forall k \notin \{s, t\}, |\sum f_{in} - \sum f_{out}| < \epsilon$.
3. **Net Flow Consistency:** Net outflow at source must equal net inflow at sink.

H.4 Diagnostic Output Examples

To illustrate how the pipeline detects errors, we present raw evaluation logs from two models on the same test instance on claude(Figure 13) and kimi(Figure 14).

Table 9: **Catalog of Evaluated Models.** Models marked with † are fine-tuned versions contributed by this work. The "VLM" column indicates native visual understanding support.

Type	Model Family	VLM	Specific Versions Evaluated
Proprietary API	OpenAI GPT	✓	`gpt-5`
	Anthropic Claude	✓	`claude-sonnet-4`
	Google Gemini	✓	`gemini-2.5-flash`
	xAI Grok	✓	`grok-4`
	DeepSeek	×	`deepseek-v3-1-terminus`
	Zhipu GLM	×	`glm-4.6`
	Moonshot Kimi	×	`kimi-k2-250905`
	ByteDance Doubao	✓	`doubao-seed-1-6`
Open-Weight	**Qwen-VL**	✓	`qwen2.5-vl-3b`, `qwen2.5-vl-3b-ft`†
			`qwen3-vl-2b`, `qwen3-vl-2b-ft`†
			`qwen3-vl-4b`, `qwen3-vl-4b-ft`†
	Qwen (Text)	×	`qwen2.5-1.5b`, `qwen2.5-1.5b-ft`†
			`qwen2.5-3b`, `qwen2.5-3b-ft`†
			`qwen3-0.6b`, `qwen3-0.6b-ft`†
			`qwen3-1.7b`, `qwen3-1.7b-ft`†
	InternVL	✓	`internvl-2b`, `internvl-4b`
			`internvl2-1b/2b/4b`, `internvl2.5-8b`
	LLaVA	✓	`llava-next-8b`

Listing F.1: The Unified User Prompt Template

[For VLMs only: Insert Image Token Here]

Your task is to solve a maximum flow problem in a directed graph defined by cities and edges with capacities.
You will output 4 lines of NDJSON (one JSON object per line). NO extra text.

CRITICAL MAPPING RULE (MANDATORY):
- You receive the cities list as the FIRST line of the user content. Treat its order as canonical.
- Map city i to id i exactly: `name_to_id[cities[0]]=0`, `name_to_id[cities[1]]=1`, ...
- Do NOT sort or reorder; do NOT infer or drop cities; use every city in that list.
- From line 2 onward, you MUST use ONLY integer ids (no city names).

Line 1 ("section":"MAP"):

```
{
  "section": "MAP",
  "rule": "by_city_mention_order",
  "name_to_id": { "<City0>":0, "<City1>":1, ... }
   contiguous 0..N-1
}
```

Line 2 ("section":"GRAPH"):

```
{
  "section": "GRAPH",
  "N": <int>,                                  // MUST equal len(cities)
  "E": <int>,                                  // number of edges
  "edges_index": [{"u":u,"v":v,"capacity":c}, ...], // ALL edges, ids only
  "source_id": <int>,                          // in [0, N-1]
  "sink_id": <int>                             // in [0, N-1] and != source_id
}
```

Line 3 ("section":"AMPL"):

```
{
  "section": "AMPL",
  "mod": "<AMPL model text using only indices...>",
  "dat": "<AMPL data using ONLY indices; set NODES... set EDGES...>"
}
```

Line 4 ("section":"SOL"):

```
{
  "section": "SOL",
  "method": "direct",
  "feasible": true|false|null,
  "objective_value": <float|null>,
  "flows_index": [{"u":u,"v":v,"f":f}, ...]    // ids only; include ALL nonzero flow
}
```

Hard rules:
- Exactly 4 JSON lines. No Markdown fences, no explanations.
- From line 2 onward: ABSOLUTELY NO city names; ids only.
- GRAPH and SOL must be consistent with the MAP ids.
- Do not invent nodes/edges/capacities not present in the input.
- Flows must be on existing edges and within capacities.

[Input Data Injection]
{{ CITIES_LIST }}
{{ PROBLEM_DESCRIPTION (Source → Sink) }}
{{ EDGE_DESCRIPTION_LIST }}

Figure 12: The standardized prompt template used for zero-shot and fine-tuning experiments.

Listing H.1: Diagnostic Log for Claude-Sonnet-4 (Instance 1)


```
{
  "metrics": {
    "GRAPH": { "edges_F1": 0.523, "graph_score": 0.504 },
    "SOL": {
      "feasible": false,
      "pred_obj": 32.0,
      "oracle_obj": 34.0,
      "objective_gap": -0.0588,
      "violations": [
        "capacity violation on (0,1): 16.0 > 13.0",
        "flow on non-existent edge (0,4)=10.0",
        "conservation violated at node 0: in=0, out=32.0"
      ]
    }
  }
}
```


Figure 13: Example of a failed solution by Claude-Sonnet-4. The metrics pipeline detected capacity violations and hallucinations of non-existent edges, automatically marking the solution as infeasible.

Listing H.2: Diagnostic Log for Kimi-k2 (Instance 2)


```
{
  "metrics": {
    "GRAPH": { "edges_F1": 0.258, "graph_score": 0.504 },
    "SOL": {
      "feasible": false,
      "pred_obj": 33.0,
      "oracle_obj": 34.0,
      "violations": [
        "flow on non-existent edge (5,4)=17.0",
        "capacity violation on (7,4): 14.0 > 5.0",
        "source/sink net mismatch: net_s=-46.0, net_t=0"
      ]
    }
  }
}
```


Figure 14: Example of a failed solution by Kimi-k2. The low Graph Score (F1=0.258) indicates poor topological understanding, leading to severe conservation violations downstream.