

VISTA: Structure-Aware Training-Free Editable Diagram Generation from Natural Language

Dongqi Han^{1*} Bojie Lin¹ Junhan Wang¹ Jiefu Luo¹

¹Beijing University of Posts and Telecommunications, Beijing, China

handongqi@bupt.edu.cn, {2025213809, 2025213817, 2025213800}@bupt.cn

Abstract

Editable diagram generation from natural language requires structural correctness and visual readability. Text-to-image models produce rasters with no editable objects. LLM code-generation approaches yield editable output but face three challenges: unreliable geometry from token-by-token coordinate emission, shallow verification that checks code rather than the rendered image, and dependence on task-specific fine-tuning. We present VISTA, a training-free pipeline that assigns each subproblem to the appropriate component: an LLM extracts semantic structure, a deterministic algorithm computes all geometry, and a VLM inspects the rendered diagram jointly with its source and issues localized repairs. We construct a 150-prompt benchmark from PNG-only references and evaluate against eight baselines including Codex, Claude Code, and a fine-tuned multi-agent system. VISTA achieves the highest readability across all pairwise comparisons. Code and data are available at <https://github.com/nice-grp/Vista>.

1 Introduction

Modern text-to-image systems, from the latent-diffusion family [Rombach et al., 2022] to Nano Banana Pro and GPT-Image-2, produce visually rich content from natural language and have become default tools for open-ended visual synthesis. Diagrams¹ are a poor fit for this paradigm. A diagram is a discrete graph of nodes, edges, and containment embedded in a continuous spatial layout, whereas pixel-based generators model only a visual distribution over the continuous space. They have no representation that enforces which boxes

Table 1: Capabilities of different text-to-diagram approaches. **Edit.**: editable output. **TF**: training-free. **Spatial**: geometry from a deterministic algorithm. **Semantic**: rendered output verified by a VLM.

Method	Edit.	TF	Spatial	Semantic
T2I diffusion	×	✓	N/A	×
LLM prompted	✓	✓	×	×
Agentic code gen.	✓	✓	×	×
AutomaTikZ	✓	×	×	×
DiagramAgent	✓	×	×	○
EvoDiagram	✓	✓	×	✓
VISTA (ours)	✓	✓	✓	✓

N/A: a raster generator has no diagram source in which geometry could be computed, so the spatial criterion does not arise. ○ Partial: Check Agent compiles the generated code and then verifies its completeness with GPT-4o, but the verification input is the code, not the rendered image.

are connected, which label belongs to which node, or whether two arrows cross by accident. The output is also a raster: flat pixels with no addressable objects, so any change requires regenerating the whole image rather than moving a node or rewiring a connection.

A diagram is, however, naturally expressible as code. Vector formats such as TikZ, Graphviz DOT [Gansner and North, 2000], Mermaid, and DrawIO XML describe a diagram as a list of typed objects—the discrete structure that the pixel representation hides. Recent work generates such code directly: by fine-tuning a code model on paired data [Belouadi et al., 2024, Ge et al., 2025, Tan et al., 2025], by orchestrating multi-agent code pipelines [Wei et al., 2025], or by letting a VLM author and refine a canvas schema [Wang et al., 2026, Zhu et al., 2026]. The output becomes editable, but three challenges persist (Table 1). (1) *No spatial guarantee*: the model must commit to numeric coordinates token by token, and LLMs are weak at exact spatial reasoning [Belouadi et al., 2024], yielding overlapping nodes and mis-routed edges; no existing system ensures collision-free ge-

* Corresponding author.

¹We use “diagrams” to denote schematic figures whose content is fully specified by a finite set of typed components (nodes, edges, containers) and explicit relations between them. This includes flowcharts, architecture diagrams, organizational charts, decision trees, and state machines, but excludes freeform illustrations and natural-image generation.

ometry by construction. (2) *No semantic guarantee on the rendered output*: where verification exists, it inspects code or compilability rather than the rendered image, so visual defects that are invisible to text-level checks survive the loop. (3) *Training dependency*: the strongest systems rely on task-specific fine-tuning, limiting generalization to new diagram types.

These challenges arise because semantic abstraction, geometric initialization, and visual repair demand different capabilities. We propose VISTA, a training-free pipeline that assigns each subproblem to the component best suited for it. An LLM extracts semantic structure and high-level *layout intent* without committing to numeric coordinates; a deterministic algorithm computes an initial layout from this structured representation; and a VLM inspects the rendered diagram jointly with its editable source to detect and repair visual defects. The deterministic stage supplies a stable, constraint-based spatial prior that makes subsequent correction more localized, while rendered-image feedback is the main driver of the final readability improvements. Here, training-free means that the system requires neither task-specific paired training data nor parameter updates; it does not imply lower inference cost.

We further build a 150-prompt benchmark spanning flowcharts, system architectures, and organizational charts, recovered from PNG-only references via VLM-based prompt extraction and manually verified. We evaluate with three complementary lenses that require no aligned DrawIO or Excalidraw ground truth: auxiliary image-level similarity, pairwise VLM readability judgment with blinded human validation, and fine-grained manual error counting over six defect indicators.

Contributions.

- A training-free framework for editable diagram generation that strictly separates semantic extraction, deterministic geometric layout, and VLM-based visual repair, providing both spatial and semantic guarantees without task-specific training.
- A 150-prompt benchmark for text-to-diagram generation, constructed from PNG-only references via VLM-based prompt recovery, together with a three-lens evaluation protocol combining image similarity, pairwise VLM readability judgment, and fine-grained manual

error counting that operates without structural ground truth.

- An empirical study against eight external baselines spanning prompted LLMs, agentic coding assistants, and a fine-tuned multi-agent system, with ablations isolating the contribution of each pipeline stage. VISTA produces the most readable diagrams overall, and the visual-feedback stage emerges as the dominant contributor to readability.

2 Related Work

Diffusion-based diagram generation. Text-to-image diffusion models [Rombach et al., 2022] produce high-quality rasters but no addressable objects or structural primitives, making editing impractical. DiagrammerGPT [Zala et al., 2024] has an LLM emit a symbolic plan that conditions a layout-guided diffusion model, but the output remains a pixel image and structural fidelity drops sharply at the rendering stage. PaperBanana [Zhu et al., 2026] orchestrates five VLM-driven agents through iterative rendering and self-critique, yet still produces raster images with faithfulness gaps such as misaligned arrows. Neither approach yields native editable objects or incorporates a deterministic, constraint-based spatial prior.

End-to-end LLM code generation. A second line treats diagram creation as code generation, building on established techniques for producing structured outputs from natural language [Zhong et al., 2017, Yin and Neubig, 2018]. AutomaTikZ [Belouadi et al., 2024] fine-tunes LLaMA on 120k TikZ-caption pairs; semantics, geometry, and styling are entangled in a single pass, leading to overlapping shapes on complex inputs, with verification limited to compilability. DiagramAgent [Wei et al., 2025] improves on this with a multi-agent framework generating TikZ or Graphviz [Gansner and North, 2000] DOT with compiler-based debugging and GPT-4o verification, but layout quality still depends on LLM-emitted coordinates and the system requires fine-tuning a 7B model. AutoPresent [Ge et al., 2025] trains an 8B model to emit Python for PPTX slides, demonstrating programmatic generation’s advantages but limited to presentations. SketchAgent [Tan et al., 2025] converts hand-drawn inputs to TikZ through a three-agent pipeline fine-tuned on sketch data, but likewise relies on LLM-generated coordinates

without a deterministic layout stage. Across these systems, geometry and semantics are handled by the same model, no typed intermediate representation decouples spatial decisions from semantic ones, and verification—where present—inspects code rather than the rendered image.

Agentic multi-stage systems. Recent work decomposes diagram generation into specialized sub-tasks. EvoDiagram [Wang et al., 2026] coordinates four agents (structure, style, layout, refiner) with an evolving design-knowledge memory, rendering through `tldraw` to achieve editable output. However, all coordinates are placed by a VLM rather than a deterministic engine, and its own ablation confirms that removing the refiner causes node overlaps—the failure mode that algorithmic layout prevents by construction. It also does not establish a source-level visual feedback loop targeting diagram-specific readability defects such as edge crossings and arrows penetrating containers. AutoFigure-Edit [Lin et al., 2026] generates editable SVG from scientific text via style-conditioned raster generation followed by VLM-based conversion, but targets illustration aesthetics rather than structured-diagram readability and couples layout quality to the initial raster stage without deterministic algorithms or rendered-image feedback. In contrast, VISTA computes geometry deterministically and closes a visual feedback loop where the VLM inspects the rendered PNG jointly with the editable source, catching defects that symbolic-plan inspection alone cannot detect.

Scientific-figure and vector-graphics generation. OCR-VQGAN [Rodríguez et al., 2023] focuses on preserving text and graphic structure during raster figure reconstruction, while FigGen [Rodríguez et al., 2023] uses diffusion-based generation for raster scientific figures. StarVector [Rodríguez et al., 2025] directly generates general SVG code and primitives from images or text, and VectorGym [Rodríguez et al., 2026] evaluates general SVG generation, sketch conversion, editing, and captioning. In contrast, VISTA targets diagrams with explicit node, edge, and container relations, computes geometry through a deterministic layout stage, and performs localized repair using both the rendered image and the native editable source.

3 Method

3.1 Overview

As shown in Figure 1, VISTA produces an editable diagram from a natural-language prompt in four stages corresponding to Sections 3.2–3.5: (i) semantic structure extraction, (ii) layout-intent planning, (iii) deterministic geometry computation, serialization, and rendering, and (iv) multimodal verification with localized source-level repair. The pipeline rests on a division of labor. Language models excel at semantic abstraction but degrade when forced to produce precise coordinates. Classical layout algorithms are reliable on geometry but blind to semantics. We therefore assign to the LLM only semantic extraction (nodes, edges, hierarchy, containers, and layout intent), delegate coordinate computation to a deterministic algorithm, and restrict the VLM to inspecting and patching the rendered artifact. This separation is what allows VISTA to be training-free and what bounds each component’s failure modes.

3.2 Semantic Structure Extraction

Given a natural-language description x , an LLM produces a structured intermediate representation $z = (V, E, H, C, L)$ as a JSON object, where V is the set of nodes (each with a semantic role and label), E is the set of directed edges, H encodes hierarchical containment, C specifies grouping containers, and L encodes layout-related attributes (Section 3.3). The prompt specifies a closed schema with typed slots for each component, so that the output can be validated against the schema rather than parsed heuristically. The layout component L is deliberately *not* a set of coordinates but a structured set of *layout hints* that encode spatial intent at the conceptual level; the model is never asked to commit to a position.

Two layers of consistency checks are applied at this stage. **Format checks** ensure that the JSON is parseable and conforms to the schema. **Reference checks** ensure that every edge endpoint refers to a declared node, that every container child is a declared node, and that hierarchical relations form a forest. If either layer fails, the model is reprompted with the offending fragment and a corrective instruction; outputs that still fail after a fixed budget are rejected. We emphasize that these are lightweight, deterministic consistency procedures rather than a strong rule-based structure verifier; defects that arise from misunderstanding the de-

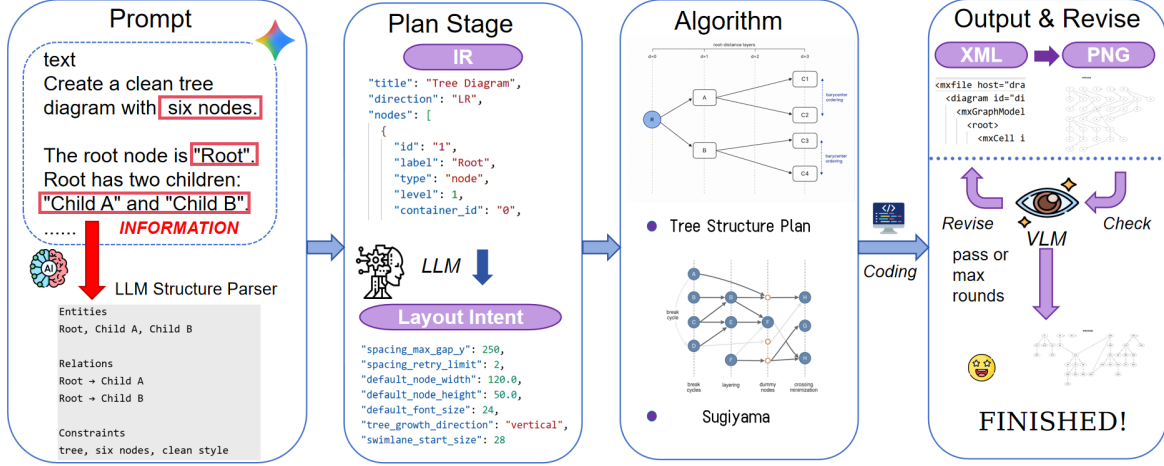


Figure 1: Overview of the VISTA pipeline. An LLM extracts semantic structure and layout intent; a deterministic algorithm computes geometry; the result is serialized to an editable source and rendered to PNG; a VLM inspects the rendered diagram and issues source-level corrections.

scription (omitted nodes, wrong relations, missing groups) are deferred to the multimodal stage in Section 3.5.

3.3 Layout Intent

Even after V , E , H , and C are fixed, the diagram is far from determined: most semantically equivalent diagrams differ primarily in their visual organization. To bridge semantics and geometry, we introduce *layout intent*, a typed taxonomy of spatial decisions. We distinguish five categories (Table 2).

Each category primarily governs a different geometric degree of freedom and is consumed by a distinct component of the layout algorithm (Section 3.4). In the implementation, layout intent is materialized as structured JSON fields (e.g., `flow_direction`, `groups`, `order_within`, `align`, `routing_style`) rather than free text.

3.4 Algorithmic Layout

Given z with populated L , a deterministic algorithm assigns coordinates to every node in V and routes every edge in E . The algorithm is a constraint-driven extension of Sugiyama-style hierarchical layout [Sugiyama et al., 1981]. Direc-

tional intent fixes the primary axis along which ranks are assigned. Grouping intent translates into shared-parent constraints that enforce containment. Alignment intent produces shared-row or shared-column constraints. Ordering intent fixes the within-rank permutation, overriding the default crossing-minimization heuristic where specified. Routing intent selects an edge-routing strategy: orthogonal routing with axis-aligned bends is the default, with straight-line fallback for short edges and explicit container-avoidance constraints that prevent edges from passing through non-endpoint nodes. Containers in C are sized to bound their children with fixed padding, and hierarchy H recursively nests bounding boxes.

Within the constraints represented by the intermediate structure, the algorithm deterministically applies rank, ordering, alignment, containment, and routing rules to produce a stable initial layout. These rules reduce avoidable overlap and routing conflicts but do not guarantee that the final rendering is collision-free. Residual defects can arise from label sizing, nested containers, route congestion, or interactions not captured by the constraint system; these are addressed by the multimodal correction stage in Section 3.5. Thus, this stage provides a deterministic spatial prior that makes correction a localized patching problem rather than a regeneration problem.

The geometric layout is serialized to DrawIO XML or Excalidraw scene data. Deterministic format checks (well-formedness, unique IDs, valid edge endpoints) are applied without invoking the

Table 2: Layout-intent categories.

Category	Controls	Example
Directional	Global flow axis	LR for pipelines
Grouping	Cluster membership	Swimlane partitions
Ordering	Within-rank sequence	Phase order in a stage
Alignment	Row/column registration	Parallel branches
Routing	Edge geometry	Orthogonal paths

LLM. The output is a syntactically valid, renderable diagram source and its rasterized PNG.

3.5 Multimodal Verification and Correction

Format-level validity does not guarantee semantic or visual correctness: nodes mentioned in the prompt may have been omitted upstream, edges may connect the wrong pair, container boundaries may overlap, or labels may overflow. We delegate detection of these defects to a VLM, which receives both the rendered PNG and the underlying editable source in a single multimodal prompt. The PNG exposes visual defects invisible to text-only inspection (overlapping containers, edge crossings, occluded labels). The source exposes structural causes (wrong parent ID, missing edge declaration) that the PNG alone cannot localize.

The VLM is asked to (i) report defects in a typed list and (ii) propose localized source-level edits that we apply directly to the editable source. The corrected document is re-rendered and the loop repeats for up to two rounds. We discuss the choice of round count in Section 4.4. This stage is framed as patching rather than regeneration: the LLM never sees geometric details, the layout algorithm never makes semantic choices, and the VLM never designs a layout from scratch. The geometric structure committed in Section 3.4 is preserved across correction rounds.

4 Experiments

4.1 Setup

We evaluate VISTA on a 150-prompt benchmark spanning flowcharts, system architecture diagrams, and organizational charts, comparing against eight external baselines: prompted LLMs (Gemini-3-pro-preview, ChatGPT-5.4, DeepSeekV4-pro, Qwen3.5-plus), agentic coding pipelines (Codex-GPT-5.4, Claude Code-DeepSeekV4-pro, Claude Code (Claude Opus 4.7)),² and one task-specific fine-tuned system (DiagramAgent).³ Our experiments examine whether spatial errors remain in the evaluated prompted and agentic systems, whether the particular fine-tuned baseline tested here closes

²Because of its substantially higher per-diagram cost, this baseline is evaluated on 29 randomly sampled diagrams rather than the full benchmark. In tables it is marked *.

³VISTA is training-free, so a fine-tuned system is not a direct point of comparison; we include DiagramAgent as the strongest publicly available representative of that line of work. Conclusions drawn from it are specific to this system and are not intended to generalize to fine-tuned approaches as a class.

this gap, and how deterministic initialization and rendered-image feedback contribute to VISTA.

Data and prompts. The benchmark contains reference PNGs without aligned DrawIO XML, Excalidraw scenes, or object-level structural annotations. Its flowchart portion builds on the flowchart category of DiagramGenBenchmark after removing duplicate and relatively simple items, and we add more complex cases. We use Qwen3.5-plus as a vision-language recognizer to recover a detailed textual prompt from each reference PNG. Each recovered prompt is manually checked against its reference, and no prompt with a known recognition error is retained. Method-development decisions, including prompt wording, layout rules, and repair heuristics, were tested on an independent pilot set collected from ProcessOn that does not overlap with the 150-prompt evaluation benchmark. The evaluation benchmark was used only for final evaluation. VISTA uses Gemini-3-pro-preview for semantic extraction and multimodal visual feedback. All outputs are uniformly rendered to PNG for evaluation. Details on prompt recovery and per-baseline evaluation scope are provided in the appendix.

Editable outputs. The final artifact produced by VISTA is a native DrawIO XML document or Excalidraw scene rather than a PNG. Nodes, text labels, connectors, and containers remain individually addressable and can be moved, relabeled, or rewired in the corresponding editor. PNGs are generated only to provide a uniform rendered view for visual feedback and evaluation. We evaluate initial generation with automated feedback rather than the subjective effort required for subsequent interactive edits.

4.2 Evaluation Metrics

Auxiliary reference-conditioned image similarity. Since reference diagrams lack XML annotations, exact node- or edge-level comparison is unavailable. We therefore use two image-level metrics as auxiliary sanity checks rather than primary evidence of diagram quality. VISTA does not aim to reproduce a reference image or its exact graph structure: for the same prompt, diagrams with different geometries and structures can be equally valid. *Graph Edit Distance (GED)*. An approximate graph structure is extracted from each PNG via image processing (node regions, geometric attributes, fill intensity, and degree features). Reference and can-

didate images are matched through bipartite assignment, and the total edit cost—aggregating position, shape, fill, and degree differences plus unmatched nodes and edge-set gaps—is normalized to $[0, 1]$ [Sanfeliu and Fu, 1983, Riesen and Bunke, 2009]. *Perceptual hash*. Each image is converted to grayscale, resized to 8×8 , and thresholded by its global mean to produce a 64-bit hash. Hash similarity is the fraction of identical bits, providing a coarse check that readability gains do not come from completely departing from the reference’s visual organization [Venkatesan et al., 2000, Zauner, 2010]. When a baseline fails to produce a diagram, both metrics are set to 0 with the denominator fixed at 150, preventing overestimation from evaluating only successful outputs.

Readability via pairwise VLM judgment. For each prompt, the VISTA PNG and a baseline PNG are shown to Gemini-3-pro-preview, which judges which diagram is clearer. The prompt restricts the evaluator to four defect types: illegible text, arrows passing through boxes, unrecognizable regions, and crossing or overlapping arrows; aesthetic preference and semantic correctness are explicitly excluded. The comparison yields a three-way label: VISTA clearer, baseline clearer, or tie; missing diagrams are excluded. Although Gemini is also used within VISTA, the judging prompt receives only two anonymized diagrams and has no access to their system identities or generation and repair histories. We further validate the judge on 50 pairs randomly sampled from the 127 valid Gemini-vs.-VISTA comparisons. Two authors independently rated each pair with method names hidden and left-right order randomized, using the same three labels and readability-defect criteria. When their initial labels differed, they jointly re-examined the pair while identities remained hidden and reached a consensus label. The consensus judgments agree with the Gemini judge on 48/50 pairs. In both disagreements, the authors selected a tie where Gemini rated VISTA as losing, which provides no evidence of pro-VISTA bias in this sample.

Fine-grained readability via manual error counting. To complement the pairwise VLM judgment, we conduct a more granular readability evaluation on 29 randomly selected diagrams. We define six error indicators targeting specific readability defects: *Arrow-box overlap* (arrows passing through node boxes), *Node overlap* (nodes occluding each other), *Text occlusion* (labels covered

Table 3: Auxiliary reference-conditioned GED and perceptual-hash similarity across baselines and ablation variants. Higher is better; bold marks the true column best.

Method	Avg GED	Avg Hash
VISTA	0.4988	0.6883
DiagramAgent	0.2312	0.5781
Codex-GPT-5.4	0.5223	0.6069
Claude Code-DeepSeekV4-pro	0.5147	0.6108
Claude Code (Claude Opus 4.7)*	0.3782	0.5587
Gemini-3-pro-preview	0.4291	0.5797
ChatGPT-5.4	0.4656	0.6484
DeepSeekV4-pro	0.4821	0.5348
Qwen3.5-plus	0.3712	0.5542
Vista-only-Algorithm	0.3850	0.5189
Vista-only-Feedback	0.5208	0.5684
Vista-Short	0.4699	0.5135

*Claude Code (Claude Opus 4.7) is evaluated on 29 randomly sampled diagrams.

by other elements), *Arrow crossings* (unnecessary edge intersections), *Detached endpoints* (arrows not properly connected to target nodes), and *Unclear arrow flow* (ambiguous edge directionality). Method names were hidden and outputs were randomly shuffled during counting. A second author independently re-counted all 29 diagrams under the same blind protocol. Agreement was high: ICC(2,1) was 0.9932, exact agreement was 98.6%, and 99.4% of paired counts differed by at most one. Discrepancies were jointly re-examined under the same blinding to produce the final consensus counts. The two raters were a senior and a junior author engaged in the study as annotators rather than as developers of the system. For each diagram and indicator, we manually count the errors of every baseline. An error count x is then normalized as:

$$x' = \left(1 - \frac{x - \min}{\max - \min}\right) \times 100,$$

where min and max are taken over all baselines for the same diagram and indicator. Higher scores indicate fewer errors. Baselines that fail to generate a diagram receive a score of 0. We report the average x' over all 29 diagrams.

4.3 Main Results

Image similarity. Table 3 reports the auxiliary similarity scores. Perceptual hash places VISTA highest (0.6883), indicating that its readability gains do not require a complete departure

Table 4: Pairwise readability counts over valid pairs after excluding missing diagrams. Each row’s three counts sum to its valid-pair total (e.g., 127 for Gemini and 29 for Claude Code (Claude Opus 4.7)*).

Baseline	Higher than Vista	Lower than Vista	Equal
DiagramAgent	9	49	66
Codex-GPT-5.4	4	70	76
Claude Code-DeepSeekV4-pro	6	56	88
Claude Code (Claude Opus 4.7)*	2	9	18
Gemini-3-pro-preview	18	46	63
ChatGPT-5.4	11	83	50
DeepSeekV4-pro	5	70	59
Qwen3.5-plus	6	81	37

* Claude Code (Claude Opus 4.7) is evaluated on 29 randomly sampled diagrams.

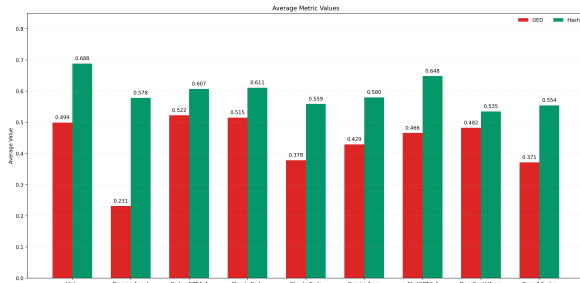


Figure 2: Auxiliary reference-conditioned GED and perceptual-hash similarity across baselines. Higher is better.

from the references’ coarse visual organization. On approximate GED, VISTA (0.4988) is competitive but not best: Codex-GPT-5.4 (0.5223), Vista-only-Feedback (0.5208), and Claude Code-DeepSeekV4-pro (0.5147) score slightly higher. Paired bootstrap analysis does not find these small GED differences significant. Because both metrics condition on a single reference even though multiple structures and geometries may be valid, they serve only as complementary checks; the readability and layout-defect evaluations provide the primary evidence. DiagramAgent, the only task-specific fine-tuned baseline evaluated here, scores 0.2312 on GED and 0.5781 on hash; this particular result shows that the tested fine-tuned system does not close the spatial-quality gap, not that fine-tuning is generally ineffective.

Readability. Table 4 shows that the aggregate comparison favors VISTA against all eight baselines. Against the strongest baselines, VISTA wins 70 comparisons vs. Codex-GPT-5.4 (which wins only 4), and 56 vs. Claude Code-DeepSeekV4-pro

(which wins 6). The most frequent ties arise against Claude Code-DeepSeekV4-pro (88 out of 150), indicating that agentic code-generation pipelines can match VISTA’s readability on a substantial proportion of inputs; nevertheless, VISTA wins nearly ten times as many decisive comparisons. The Claude Code (Claude Opus 4.7)* comparison has the same aggregate direction but does not reach statistical significance, with 18 ties among its 29 valid pairs. Against the remaining baselines the margins are larger; for example, ChatGPT-5.4 loses 83 comparisons and Qwen3.5-plus loses 81. Overall, the pairwise results favor VISTA against all evaluated baselines, while the Claude Code (Claude Opus 4.7)* result should be interpreted cautiously. Additional qualitative baseline montages are provided in Appendix F.

Computational cost. Table 5 reports token consumption and wall-clock time per diagram. Codex-GPT-5.4 is the most token-intensive baseline (mean 47,334), followed by VISTA (mean 26,862); the remaining single-call baselines range from 3,679 to 9,238 tokens. VISTA’s elevated token usage reflects its multi-stage pipeline: the initial LLM extraction takes 87.2 s on average, two VLM feedback rounds add 48.8 s and 64.3 s, and compilation repair contributes only 7.5 s (median 0 s, indicating most diagrams require no repair). Despite the higher token budget, VISTA’s wall-clock time (207.9 s) remains below DeepSeekV4-pro (258.6 s) and is comparable to Codex-GPT-5.4 (132.5 s), while the fastest baselines complete in under a minute (ChatGPT-5.4 at 51.1 s, Qwen3.5-plus at 57.7 s). Notably, VISTA, Codex-GPT-5.4, and Claude Code-DeepSeekV4-pro all successfully process all 150 prompts, whereas other baseline completion rates range from 82.7% (Qwen3.5-plus and DiagramAgent) to 96.0% (ChatGPT-5.4). The additional compute directly funds the feedback loop that produces the quality gains reported in Tables 3–4; efficiency optimization remains an important direction.

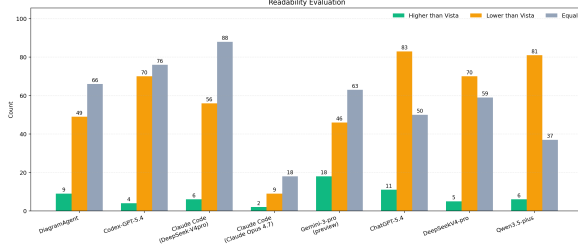


Figure 3: Pairwise comparison counts on image similarity (top two) and readability (bottom).

Table 5: Per-diagram token consumption and wall-clock time. N : number of successfully generated samples (out of 150). Baseline statistics are computed over successful samples only; VISTA includes all 150 prompts.

Method	N	Tokens		Time (s)	
		Mean	Median	Mean	Median
Codex-GPT-5.4	150	47,334	31,092	132.5	109.0
Claude Code-DeepSeekV4-pro	150	6,662	5,502	88.5	86.9
Claude Code (Claude Opus 4.7)*	29	12,663	10,919	97.3	77.0
DeepSeekV4-pro	134	9,238	8,950	258.6	222.6
DiagramAgent	124	6,488	6,001	123.5	116.2
Gemini-3-pro-preview	127	3,679	3,440	106.1	99.2
ChatGPT-5.4	144	7,848	7,714	51.1	47.3
Qwen3.5-plus	124	3,877	3,719	57.7	56.1
VISTA	150	26,862	26,335	207.9	200.0

*Claude Code (Claude Opus 4.7) is evaluated on 29 randomly sampled diagrams.

4.4 Fine-Grained Readability and Ablation

To isolate the contribution of each pipeline stage, we evaluate three ablation variants alongside the full system and external baselines using the error-based readability metrics.

Ablation variants. **Vista-Short** uses abbreviated prompts that retain only connection relationships and component styles but omit explicit layout intent (e.g., tree layout, hexagonal arrangement), testing whether high-level spatial directives in the prompt are necessary. **Vista-only-Feedback** removes the deterministic layout algorithm entirely: the LLM directly generates diagram code, and the VLM feedback loop refines the rendered output without an algorithmic initial draft. **Vista-only-Algorithm** removes the VLM feedback loop: the diagram is produced by semantic extraction and algorithmic layout alone, with no visual verification or correction.

Baseline comparison. Table 6 and Figure 4 report normalized readability scores across the six error indicators. The full VISTA pipeline achieves the highest or near-highest score on every indicator, attaining 100.00 on Node overlap, Detached endpoints, and Unclear arrow flow within this normal-

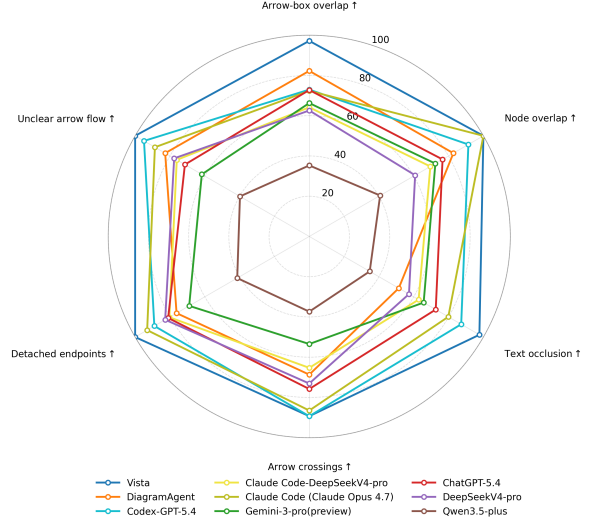


Figure 4: Radar chart of normalized readability scores. VISTA (outermost polygon) approaches the maximum on all six axes; all baselines exhibit visible concavities on at least two indicators.

ization. Among external baselines, Codex-GPT-5.4 is the strongest competitor, exceeding 85 on five of six indicators and matching VISTA on Arrow crossings (both 89.31), yet trailing on Arrow-box overlap (72.96 vs. 97.17). Claude Code (Claude Opus 4.7)* achieves strong scores on Node overlap (100.00) and Arrow crossings (86.50), but it still trails VISTA on Arrow-box overlap and Text occlusion. DiagramAgent scores 51.40 on Text occlusion and 68.66 on Arrow crossings; this shows that the particular fine-tuned baseline evaluated here does not close the observed spatial-quality gap. The radar chart visualizes the broader pattern, while absolute error totals are reported in Table 8.

Ablation analysis. The three ablation variants provide independent evidence for each pipeline component.

Vista-only-Algorithm (no VLM feedback) has 379 counted errors, compared with 25 for the full system. Its low normalized scores show that deterministic initialization alone cannot address fine-grained visual defects such as label sizing, overlap, and route congestion because the layout engine lacks pixel-level awareness. Its role is to provide a stable, constraint-based spatial prior for subsequent repair rather than to guarantee a defect-free rendering.

Vista-only-Feedback (no algorithmic layout) has 72 counted errors and maintains reasonable normalized scores, but it falls behind the full pipeline’s 25 errors. Without a stable initial spatial prior, the

Table 6: Normalized detection scores across error metrics with column-wise color gradients. Normalization is performed per diagram and indicator over the compared methods; 100 denotes the best count within that comparison and does not necessarily mean zero raw errors.

Method	Arrow-box overlap↑	Node overlap↑	Text occlusion↑	Arrow crossings↑	Detached endpoints↑	Unclear arrow flow↑
Vista	97.17	100.00	97.70	89.31	100.00	100.00
Vista-Short	98.10	100.00	99.31	88.52	95.86	100.00
Vista-only-Feedback	90.07	94.02	94.75	76.49	96.55	97.84
Vista-only-Algorithm	25.76	55.17	50.31	49.79	62.07	37.76
DiagramAgent	82.27	82.76	51.40	68.66	76.21	82.76
Codex-GPT-5.4	72.96	91.38	87.25	89.31	88.97	94.91
Claude Code-DeepSeekV4-pro	64.17	69.54	62.80	65.29	80.34	76.09
Claude Code (Claude Opus 4.7)*	72.35	100.00	79.84	86.50	93.10	88.65
Gemini-3-pro-preview	66.28	72.41	65.69	53.45	68.97	61.72
ChatGPT-5.4	72.76	76.55	72.63	75.80	81.03	71.43
DeepSeekV4-pro	62.59	60.80	57.26	73.08	82.76	77.59
Qwen3.5-plus	35.29	40.69	34.71	37.36	41.38	39.80

* Claude Code (Claude Opus 4.7) is evaluated on 29 randomly sampled diagrams.

Table 7: Normalized readability scores across feedback rounds. *Algorithm*: no VLM feedback; *Rk*: after k rounds. Normalization is within this round-ablation group, so Algorithm’s 0.0 marks the group minimum rather than zero raw errors.

Round	Arrow-box↑	Node overlap↑	Text occl.↑	Arrow cross.↑	Detached↑	Unclear flow↑
Algorithm	0.0	0.0	0.0	0.0	0.0	0.0
R1	88.0	80.0	91.7	66.2	80.0	92.0
R2	99.0	100.0	100.0	98.6	100.0	100.0
R3	100.0	100.0	100.0	100.0	100.0	100.0

VLM feedback loop must both discover and repair geometry. Together, the ablations show that rendered-image feedback is the main direct driver of readability improvement, while deterministic initialization makes that repair problem substantially easier.

Vista-Short (no explicit layout intent in the prompt) performs comparably to the full system on most indicators, with only minor drops on Arrow crossings (88.52 vs. 89.31) and Detached endpoints (95.86 vs. 100.00). This indicates that VISTA’s pipeline architecture, rather than prompt engineering, is the primary source of readability gains.

Effect of feedback rounds. Table 7 reports readability scores as a function of the number of VLM feedback rounds. Without feedback (Algorithm only), all normalized scores are at the minimum. The first round produces large gains (66–92 across indicators), but residual defects remain, particularly on Arrow crossings (66.2). The second round brings all indicators to near-perfect levels (98.6–100.0). A third round closes the remaining gap entirely, but the marginal improvement over two rounds is small (≤ 1.4 points on any indicator). We therefore set the default to two rounds, which captures the bulk of visual improvement while avoid-

ing additional cost and the risk of over-editing.

5 Conclusion

VISTA generates native editable diagrams by separating semantic extraction and layout intent from deterministic geometric initialization and rendered-image repair. Across the evaluated systems, the full pipeline achieves the highest or near-highest normalized readability scores and the aggregate pairwise direction favors VISTA against all eight baselines. The ablations identify multimodal feedback as the main direct driver of the readability gains, with deterministic geometry supplying a stable spatial prior for localized repair. These results support this division of labor within the evaluated diagram domains and baselines; training-free here means that no task-specific paired training data or parameter updates are required.

Limitations

First, the evaluation covers flowcharts, system architectures, and organizational charts. Coordinate-system-based figures (function plots, charts with continuous curves) require specialized backends and additional IR fields beyond DrawIO’s primitive set. Second, our readability evaluation uses Gemini as the scalable pairwise judge, with the generation and judging roles separated by construction so that no prompt or repair history leaks between them. How closely such automatic judgments track the preferences of external expert users remains a question for future work. Third, VISTA consumes more tokens and time than single-call baselines due to its multi-stage pipeline. Training-free means no task-specific parameter updates, not inexpensive inference. Reducing redundant visual tokens and batching feedback requests remain important engineering directions. Finally, we instantiate only two editable backends (DrawIO and Excalidraw). Sensitivity to alternative routing strategies and repair-prompt designs has not been systematically analyzed. Supporting a new target also requires a serializer, renderer, and format-specific repair rules.

Acknowledgement

This work was supported by the Fundamental Research Funds for the Central Universities and by the National Natural Science Foundation of China (Grant No. 62502043).

References

- Jonas Belouadi, Anne Lauscher, and Steffen Eger. AutomaTikZ: Text-guided synthesis of scientific vector graphics with TikZ. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.
- Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software: Practice and Experience*, 30(11):1203–1233, 2000.
- Jiaxin Ge, Zora Zhiruo Wang, Xuhui Zhou, Yi-Hao Peng, Sanjay Subramanian, Qinyue Tan, Maarten Sap, Alane Suhr, Daniel Fried, Graham Neubig, and Trevor Darrell. AutoPresent: Designing structured visuals from scratch. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2902–2911, 2025.
- Zhen Lin, Qiujie Xie, Minjun Zhu, Shichen Li, Qiyao Sun, Enhao Gu, Yiran Ding, Ke Sun, Fang Guo, Panzhong Lu, Zhiyuan Ning, Yixuan Weng, and Yue Zhang. AutoFigure-Edit: Generating editable scientific illustrations via reference-guided styling. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 57–67, 2026. doi: 10.18653/v1/2026.acl-demo.6.
- Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27(7):950–959, 2009. doi: 10.1016/j.imavis.2008.04.004.
- Juan Rodriguez, Haotian Zhang, Abhay Puri, Tianyang Zhang, Rishav Pramanik, Meng Lin, Xiaoqing Xie, Marco Terral, Darsh Kaushik, Aly Shariff, Perouz Taslakian, Spandana Gella, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. VectorGym: A multitask benchmark for svg code generation, sketching, and editing. *arXiv preprint arXiv:2603.29852*, 2026.
- Juan A. Rodriguez, David Vazquez, Issam Laradji, Marco Pedersoli, and Pau Rodriguez. FigGen: Text to scientific figure generation. In *ICLR 2023 Tiny Papers*, 2023.
- Juan A. Rodríguez, David Vazquez, Issam Laradji, Marco Pedersoli, and Pau Rodriguez. OCR-VQGAN: Taming text-within-image generation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3689–3698, 2023.
- Juan A. Rodriguez, Abhay Puri, Shubham Agarwal, Issam H. Laradji, Pau Rodriguez, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. StarVector: Generating scalable vector graphics code from images and text. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16175–16186, 2025.
- Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, 2022.
- Alberto Sanfeliu and King-Sun Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(3):353–362, 1983. doi: 10.1109/TSMC.1983.6313167.
- Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiro Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981. doi: 10.1109/TSMC.1981.4308636.
- Cheng Tan, Qi Chen, Jingxuan Wei, Gaowei Wu, Zhangyang Gao, Siyuan Li, Bihui Yu, Ruifeng Guo, and Stan Z. Li. SketchAgent: Generating structured diagrams from hand-drawn sketches. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1918–1926, 2025.

R. Venkatesan, S. M. Koon, M. H. Jakubowski, and P. Moulin. Robust image hashing. In *International Conference on Image Processing (ICIP)*, pages 664–666, 2000.

Tianfu Wang, Leilei Ding, Ziyang Tao, Yi Zhan, Zhiyuan Ma, Wei Wu, Yuxuan Lei, Yuan Feng, Junyang Wang, Yin Wu, Yizhao Xu, Hongyuan Zhu, Qi Liu, Nicholas Jing Yuan, Yanyong Zhang, and Hui Xiong. EvoDiagram: Agentic editable diagram creation via design expertise evolution. *arXiv preprint arXiv:2604.09568*, 2026.

Jingxuan Wei, Cheng Tan, Qi Chen, Gaowei Wu, Siyuan Li, Zhangyang Gao, Linzhuang Sun, Bihui Yu, and Ruifeng Guo. From words to structured visuals: A benchmark and framework for text-to-diagram generation and editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13315–13325, 2025.

Pengcheng Yin and Graham Neubig. TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 7–12, 2018.

Abhay Zala, Han Lin, Jaemin Cho, and Mohit Bansal. DiagrammerGPT: Generating open-domain, open-platform diagrams via LLM planning. In *Proceedings of the Conference on Language Modeling (COLM)*, 2024.

Christoph Zauner. Implementation and benchmarking of perceptual image hash functions. Master’s thesis, University of Applied Sciences Upper Austria, 2010.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

Dawei Zhu, Rui Meng, Yale Song, Xiyu Wei, Sujian Li, Tomas Pfister, and Jinsung Yoon. PaperBanana: Automating academic illustration for AI scientists. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026.

A Author Contributions

Dongqi Han. Dongqi Han conceived the initial research idea and contributed to shaping the overall research direction. He designed the experimental protocol and ablation studies, reviewed experimental procedures and results, contributed to the interpretation and discussion of the findings, and participated extensively in manuscript drafting, organization, revision, and final review.

Bojie Lin. Bojie Lin is the leading student author of this work. He carried the primary implementation responsibility, including developing the core technical methodology and its major methodological refinements. He implemented the complete system, constructed the benchmark and evaluation framework, designed and conducted the main experiments, performed the statistical analyses and result visualization, and contributed to manuscript drafting and revision.

Junhan Wang. Junhan Wang contributed to experimental execution, external data collection, reproduction of related methods, and experimental validation.

Jiefu Luo. Jiefu Luo contributed to experimental execution, external data collection, reproduction of related methods, and experimental validation.

B Generation Prompts

This appendix lists the main prompts used by VISTA. The prompts are written to match the division of labor in Section 3: language models extract semantic structure and layout intent, the deterministic layout engine computes final geometry, and the visual-feedback stage performs localized source repair. Prompts recovered from reference PNGs are manually checked before being used for evaluation.

B.1 Structure System Prompt

```
You are a structure extractor for DiagramAI.
Return only valid JSON.
Do not output markdown, code fences, comments, or
explanations.

Required top-level fields:
- "title": string
- "hierarchy": array
- "edges": array

Optional top-level fields:
- "style"
- "layout"

Rules:
- Keep labels faithful to the input.
- Do not invent ids.
- Do not include the title itself as a node in hierarchy.
- Use edges only when the input explicitly describes
relationships.
- Edge objects use {"from": "A", "to": "B", "link": "dir"} with
link values: dir, undir, loose.
- Prefer a simple hierarchy and minimal nesting.
- Do not assign final absolute coordinates.
- If useful, include approximate label-size hints for long
labels.
These hints are for readability and may be overridden by
the
deterministic layout engine.
```

B.2 Grid System Prompt

```
You are a layout-intent planner for DiagramAI.
Return only valid JSON.
Do not output markdown, code fences, comments, or
explanations.

Generate a simple layout-intent plan that respects the
provided
```

```
hierarchy. Prefer straightforward relative placements,
stable spacing
preferences, and low edge crossing.

Do not assign final absolute node coordinates. Provide only
semantic
layout intent, such as sequence direction, side-by-side
grouping,
alignment preference, spacing preference, and routing
preference.
The deterministic layout engine will compute final rows,
columns,
coordinates, sizes, and routes.
```

B.3 Structure User Prompt

```
Convert the input into one JSON object for DiagramAI.
Return only valid JSON with this shape:
{"title":"Diagram Title","hierarchy":[],"edges":[],
"style":{"container_scheme":"swimlane_alt","root_layout":"j
v"},
"layout":{"tree_growth_direction":"horizontal",
"gap_x":80,"gap_y":100,"margin":80}}

Rules:
- Use the highest-level heading as title when present.
- Do not include the title itself inside hierarchy.
- hierarchy may contain strings, {"Group":[...]}, or
{"node":{"label":"...", "width":N, "height":N, "fontSize":2
8}}.
- Keep labels faithful to the input.
- Add edges only when explicitly described.
- Edge format: {"from":"A","to":"B","link":"dir"};
use link values dir, undir, or loose.
- For labels longer than 4 chars, approximate
width/height/fontSize
may be included only as readability hints. The
deterministic layout
engine may override these values.
- Prefer simple hierarchy. Use containers only when the
input clearly
groups items.
- Do not assign final absolute coordinates.
- No markdown, no code fences, no explanation.

Input:
"""
{input_text}
"""
```

B.4 Grid Planning User Prompt

```
Create a layout-intent JSON for the provided DiagramAI
context.
Return only valid JSON.
Required shape:
{"containers":{},"layout":{"gap_x":80,"gap_y":100,"margin"
:80}}

Rules:
- Respect hierarchy and container intent from the context.
- For linear flows, specify one preferred flow direction and
avoid
crossing-prone intent.
- For side-by-side groups, specify relative grouping intent;
for
sequences, specify ordering intent.
- Keep placements simple and consistent.
- Include per-container intent only when needed.
- Do not assign final absolute coordinates. The
deterministic layout
engine will compute final rows, columns, coordinates,
spacing, and
routes.
- No markdown, no code fences, no explanation.

Context:
"""
{context_json}
"""
```

B.5 Refinement Prompt

```
Patch the existing diagram structure as valid JSON.
Apply the feedback to improve layout clarity, spacing, and
edge
readability.
Return only one JSON object with fields: title, hierarchy,
edges,
optional style, optional layout.
Preserve the original structure and only revise layout
intent or local
```

```
readability-related fields.
Do not include markdown or explanation.
```

```
Original input:
"""
{original_prompt}
"""

Feedback:
"""
{gemini_feedback}
"""
```

B.6 Enhanced Refinement Extra Constraints

[Key Constraints – Must Follow]

1. Preserve structural completeness. Do not delete any nodes or change the basic flow logic.
2. Keep sizes reasonable. Size values are readability hints; do not make nodes too large or too small, and keep them within +/-20% of the original hint when such a hint exists.
3. Preserve colors. If the original design has specific colors, keep them consistent.
4. JSON format must be valid. The final JSON must be parseable.
5. Only fix the identified problems. Do not make unrelated improvements.
6. For directed graphs without explicit grouping or swimlanes, keep nodes flat at the top level of hierarchy. Use containers only when the input explicitly describes groups, swimlanes, modules, or nested scopes.
7. Do not assign final absolute coordinates. Provide only structure and layout intent; the deterministic layout engine computes final positions, sizes, and routes.

[Quality Checklist]

Before modifying, confirm:

- All node names and IDs remain unchanged.
- All connection relationships remain correct.
- The JSON structure is still valid.
- The modification only addresses the reported issues.
- For diagrams with edges and no explicit groups, hierarchy contains no nested containers.

B.7 Gemini Image Layout Review Prompt

You are a diagram layout quality-check assistant. Use the diagram-generation prompt and the provided PNG image to analyze layout problems and provide detailed revision suggestions.

Important: Your issues and suggestions will be sent to another Gemini model for automatic diagram improvement. Make your feedback detailed, specific, and actionable. If the layout is already clear enough, do not over-analyze or propose unnecessary changes.

Key checks:

1. Missing or incorrect elements: missing nodes, connectors, labels, wrong labels, or incorrect connections.
2. Node overlap: boxes overlap, touch edges, exceed the view boundary, or text overlaps.
3. Arrow crossing: unnecessary connector crossings; identify which arrows cross and where.
4. Arrow through box: arrows passing through non-endpoint nodes.
5. Text fitting: text too large, outside node boundaries, or overlapping other elements/arrows.

Core principle: prioritize fixing layout by changing relative placement intent and layout parameters. Do not assign final absolute coordinates in this review stage.

For each issue, first consider:

- changing flow direction, ordering, grouping, alignment, gap_x, gap_y, margin, root_layout, container layout direction, canvas size, or component translation intent;
- changing arrow style, curvature, connection points, or line type;
- changing hierarchy or grouping only when needed.

Only consider other solutions when position planning and layout parameters cannot solve the problem.

Output requirements:

1. List all layout problems, especially node overlap, arrow crossing, and arrow-through-box problems.
2. For each issue, provide solutions in this priority order:
 - Priority 1: relative layout-intent replanning, including hierarchy or container changes and an example JSON structure.
 - Priority 2: layout parameter changes, such as gap_x, gap_y, margin, root_layout.
 - Priority 3: other changes, such as routing, color, or labels.
3. Sort issues by user-experience impact.
4. Do not restate the image content or the generation prompt.
5. If no obvious layout issue is found, output:
 - No obvious layout issues found.

Diagram generation prompt:

```
{diagram_prompt}
```

B.8 Gemini Analyze-And-Fix DrawIO Prompt

You are a diagram layout quality-check assistant, feedback round {round_num}.

Task:

Use the PNG image and the DrawIO XML source below to analyze diagram layout problems and directly output the complete corrected DrawIO XML.

User diagram-generation prompt:

```
{diagram_prompt}
```

Current DrawIO XML source:

```
{drawio_code}
```

Checkpoints:

1. Node overlap: any box physically overlaps another box.
2. Arrow crossing: unnecessary connector crossings.
3. Arrow through box: an arrow passes through a non-endpoint node.

Output format requirements:

Part 1: Revision suggestions

List the detected problems and concise modification notes.

If no obvious layout issue is found, output:

No obvious layout issues found. In that case, do not output Part 2.

---CORRECTED_CODE---

Part 2: Complete corrected DrawIO XML

Output the corrected complete DrawIO XML directly, without wrapping it in ```xml.

Only adjust node coordinates (x, y) and sizes (width, height) to solve layout problems. Keep node structure, connector relationships, and styles unchanged.

The XML must be complete and directly saveable as a .drawio file.

B.9 DrawIO Syntax Repair Prompt

The previously generated DrawIO XML contains illegal syntax or unsupported node structures, causing the draw.io renderer to fail and PNG export to crash.

Because export failed, no image is available.

Please inspect the original requirement and the crashing XML code below, and fix XML tag or syntax errors. For example, do not use illegal attributes such as <Array points="...">; instead use valid draw.io syntax such as <Array as="points"><mxPoint x="..." y="..."></Array>.

Original requirement:

```
{diagram_prompt}
```

DrawIO XML code that caused the rendering crash:

```
{drawio_code}
```

Output the analysis and corrected complete code in this format:

Part 1: Analysis

Explain the XML syntax/tag-format problem that caused the crash and briefly describe the fix.

---CORRECTED_CODE---

Part 2: Complete corrected DrawIO XML

Directly output the complete corrected DrawIO XML without wrapping it in ```xml.

It must be complete XML that can be opened or exported by draw.io. Keep all correct coordinates and structure unchanged; only fix syntax errors.

B.10 Feedback Issue Extraction Prompt

Please analyze the following feedback and extract all layout issues.

Use this JSON format. Return JSON only, with no other text: [{ "id": 1, "type": "issue type", "description": "short description" }]

If there are no issues, return an empty array [].

Feedback:

```
{feedback_text}
```

B.11 Issue History Comparison Prompt

You are an issue analysis expert. Compare the issues found in the current round, round {current_round}, with issues from previous rounds, and determine whether new issues have appeared.

Core task: determine whether all current issues are repeated issues.

Criteria:

1. Repeated issue: if the core content of the current issue, meaning issue type plus affected area, already appeared in historical issues, treat it as repeated.
2. New issue: treat it as new if it involves different nodes/areas, a completely different issue type, or a clearly different severity.
3. Lenient rule: if uncertain, prefer judging it as a new issue so improvement continues. Only mark it repeated when it is clearly repeated.

Current round issues:

```
{current_issues}
```

Historical issues:

```
{historical_issues}
```

Output the analysis result as JSON only:

```
{
  "has_new_issues": true or false,
  "new_issue_count": number of new issues,
  "repeated_issue_count": number of repeated issues,
  "analysis": "brief explanation in 1-2 sentences"
}
```

Logic:

- If new_issue_count > 0, has_new_issues = true.
- If all current issues are repeated, has_new_issues = false.

B.12 Multi-Round Trend Analysis Prompt

You are a diagram quality analysis expert. Analyze the following multi-round Gemini feedback and summarize improvement trends and persistent issues.

Total rounds: {total_rounds}
Termination reason: {termination_reason}

Feedback from each round:
{all_feedbacks_text}

Please analyze:

1. The improvement trend across multiple revisions: are issues decreasing, and which issues were resolved?
2. Persistent issues: which issues remain unresolved?
3. Possible root causes: why are some issues difficult to solve?
4. Suggested solution direction: how to improve the diagram fundamentally?

Output a concise analysis report of 200-300 Chinese characters.

C Qwen3.6Max VLM Example

To examine backend dependence, we additionally run the pipeline with Qwen3.6Max in place of Gemini-3-pro-preview while keeping every other component (semantic schema, deterministic layout, and source-level edit loop) unchanged. On the same 29 diagrams and six indicators, Vista-Qwen3.6 obtains normalized scores of 90.98, 99.43, 92.59, 89.43, 100.00, and 98.28, respectively, with 55 total counted errors. This is above standard VISTA’s 25 errors but far below standalone Gemini-3-pro-preview’s 250 and Qwen3.5-plus’s 371 under the same rules. The result indicates that the pipeline remains effective with another capable backend, although the Gemini-backed configuration performs better. Figure 5 shows five representative outputs.

D Excalidraw Output Example

Figure 6 shows representative diagrams generated by VISTA through the Excalidraw backend. This example illustrates that the intermediate representation and correction loop are not tied to DrawIO alone: the same semantic and layout structure can be serialized into an editable Excalidraw scene, rendered for visual inspection, and revised through source-level edits.

E Iterative Refinement Progression

Figure 7 provides a qualitative view of how VISTA’s multimodal verification and correction loop reshapes the same diagram across rounds. Each column corresponds to one prompt, the top row (*vista-round1*) shows the initial draft produced after a single pass of semantic extraction and deterministic layout, and the bottom row (*vista-round3*) shows the result after two additional rounds of

Table 8: Final consensus absolute error counts on the 29-diagram fine-grained readability subset. Lower is better. A–F correspond to Arrow-box overlap, Node overlap, Text occlusion, Arrow crossings, Detached endpoints, and Unclear arrow flow.

Method	A	B	C	D	E	F	Total
VISTA	5	0	2	18	0	0	25
Vista-Short	3	0	1	22	3	0	29
Vista-only-Feedback	14	5	7	41	2	3	72
Codex-GPT-5.4	37	3	18	23	7	5	93
Claude Code (Claude Opus 4.7)*	38	0	29	30	7	21	125
DeepSeekV4-pro	38	17	53	50	2	16	176
ChatGPT-5.4	40	15	38	47	13	29	182
DiagramAgent	32	9	84	79	11	18	233
Claude Code-DeepSeekV4-pro	48	18	46	83	10	29	234
Gemini-3-pro-preview	42	12	59	89	15	33	250
Qwen3.5-plus	82	22	89	108	20	50	371
Vista-only-Algorithm	104	30	62	88	20	75	379

*Claude Code (Claude Opus 4.7) is evaluated on 29 randomly sampled diagrams.

source-level edits driven by VLM visual feedback. The contrast highlights the kinds of failures the loop is designed to repair: cluttered or grid-aligned arrows are reorganized into routed connectors with clear direction, overlapping nodes are separated, decision branches acquire a consistent top-to-bottom or left-to-right flow, and labels that were truncated or partially occluded become legible. These per-diagram trajectories indicate that the gains reported in Section 4 are not the product of a single one-shot generation but of the verify-and-fix loop progressively closing the gap between what the model can perceive and what raw layout heuristics can produce.

F Baseline Montage Comparison

Figure 8 compares VISTA (top row, intentionally unlabeled) with eight external baselines on five selected benchmark prompts. Each row corresponds to one method and each column to one target prompt, enabling direct vertical comparison of different systems on the same input. The baselines, from the second row downward, are Codex-GPT-5.4, Claude Code-DeepSeekV4-pro, DeepSeek-v4-pro, DiagramAgent, Gemini-3-pro-preview, ChatGPT-5.4, Qwen-3.5-plus, and Claude Code (Claude Opus 4.7)*. Cells marked with a black cross indicate that the corresponding system either failed to return a valid diagram or did not produce an output for that prompt; this pattern is most visible for Qwen-3.5-plus and Claude Code (Claude Opus 4.7)*, both of which have fewer usable outputs on this slice. The montage makes the recurring failure modes targeted by VISTA easy

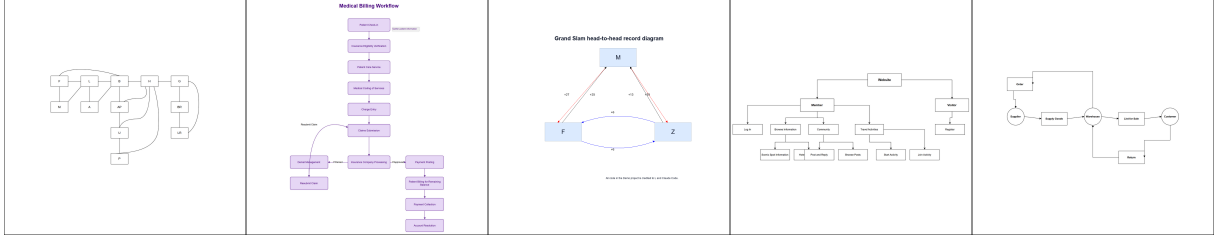


Figure 5: Representative VISTA outputs produced with Qwen3.6Max as the VLM for semantic extraction and visual feedback, with all other components unchanged. Each panel is rendered from an editable diagram source.

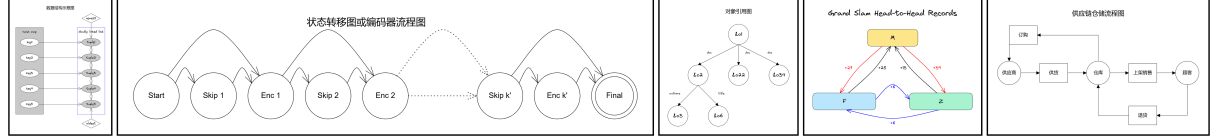


Figure 6: Representative VISTA outputs generated with the Excalidraw backend. Each panel is rendered from an editable Excalidraw scene rather than a flat raster-only image.

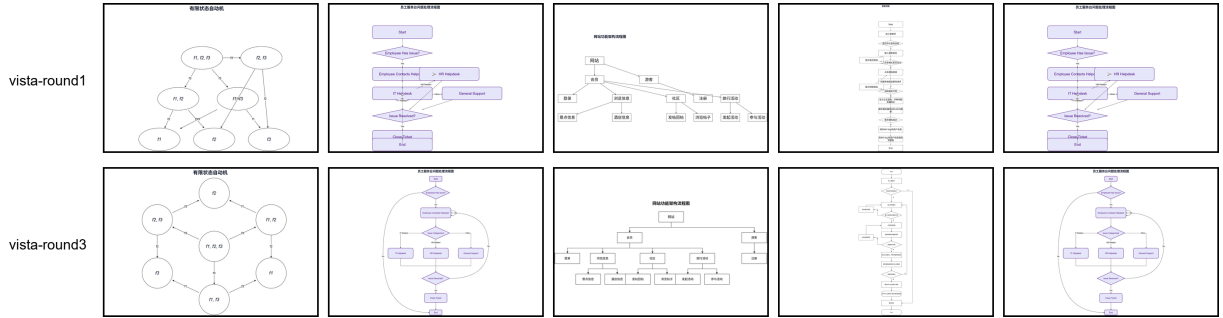


Figure 7: Round-by-round refinement of VISTA. Each column is the same diagram prompt; the top row shows the round-1 output (before any visual feedback), and the bottom row shows the round-3 output, obtained after two rounds of multimodal verification and source-level correction.

to see at a glance: tiny or unreadable labels, overlapping connectors, arrows that pass through node interiors, and decision branches that collapse into a vertical or diagonal pile without a clear top-to-bottom or left-to-right flow.

G Code and Data Availability

The complete source code of VISTA, including the semantic extraction module, deterministic layout engine, VLM feedback loop, and rendering pipeline, is available at <https://github.com/nice-grp/Vista>. The same repository hosts the benchmark: the 150 recovered prompts, together with provenance pointers to their source diagrams. Reference images whose original licensing terms do not permit redistribution are identified by pointer rather than redistributed. Details on data collection, prompt recovery, and manual verification are described in Section 4 and Appendix B.

H Baseline Evaluation Scope

Seven of the eight external baselines are evaluated on the full 150-prompt benchmark. Claude Code (Claude Opus 4.7)*, an agentic coding pipeline built on a frontier model, was evaluated on a randomly sampled subset of 29 prompts due to its substantially higher per-diagram computational cost. The 29 prompts were drawn uniformly from the same pool used by all other baselines, covering the same distribution of flowcharts, system architecture diagrams, and organizational charts. All metrics reported for Claude Code (Claude Opus 4.7)* in the main text are computed on this subset.

Vista					
Codex-GPT-5.4					
Claude Code-DeepSeekV4-pro					
Deepseek-v4pro					
DiagramAgent					
Gemini-3-pro(preview)					
ChatGPT-5.4					
Qwen-3.5plus					
Claude Code(Claude Opus 4.7)					

Figure 8: Baseline montage on selected benchmark prompts. The top row shows VISTA (intentionally unlabeled) and the remaining eight rows show external baselines; each column corresponds to one target prompt. Black crosses indicate missing or failed outputs.