ProGraf: Profile-Guided Planning for Step-by-Step Generation of Structured Non-Natural Images

Ran Luo[1] **Haoxiang Deng**[1] **Lan Zhang**[1,2*] **Mu Yuan**[3]
[1]University of Science and Technology of China
[2]Institute of Artificial Intelligence,
Hefei Comprehensive National Science Center
[3]The Chinese University of Hong Kong
luoran123@mail.ustc.edu.cn, hx.deng@mail.ustc.edu.cn
zhanglan@ustc.edu.cn, muyuan@cuhk.edu.hk

Abstract

Structured visual artifacts such as geometry diagrams, circuit schematics, and flowcharts are increasingly important targets for image generation, yet visual plausibility alone is insufficient—a single missing object, wrong relation, broken connection, or extra element can invalidate the output. Existing image generators remain brittle on such tasks, and generic step-by-step or agentic methods often fail because intermediate edits may corrupt previously correct structures and ignore generator-specific failure modes. To address this, we present ProGraf, a profile-guided framework for recovering structured non-natural image generation failures with black-box raster generators. We first build a benchmark of 304 tasks across geometry, circuit, flowchart, and general diagrams, each annotated with fine-grained verification points. From repeated generator outputs, we derive model-specific capability profiles capturing category-level reliability, component-level weaknesses, and recurrent failure patterns. ProGraf uses these profiles to guide closed-loop decomposition, verifies each intermediate result, commits only accepted states, and replans after failures. On generator-specific hardest subsets, ProGraf recovers 43.7–77.8% of previously failed tasks across generators and categories, outperforming fixed-step and adapted agent baselines; ablation studies further show that benchmark-derived profiles are a key source of the gain.

1 Introduction

Recent advances in image generation have substantially improved open-ended visual synthesis [21, 23, 24, 26, 42], yet structured non-natural image generation remains brittle. Unlike natural image prompts, tasks such as geometric diagrams, circuit schematics, flowcharts, and abstract graphs require models to satisfy explicit object identities, relation constraints, and negative constraints simultaneously [7–9, 14, 22, 31]. A result may look visually plausible while still being incorrect because of missing elements, broken topology, wrong geometric relations, or extra unwanted structures [7, 9, 14, 27, 31].

A natural strategy is to decompose a difficult task into multiple generation or editing steps [4, 10, 15, 16, 19, 28, 32]. Recent agentic systems further use structured planning for complex editing [36] or closed-loop multi-agent refinement for interactive generation and editing [18]. However, decomposition alone does not resolve structured generation: intermediate edits can damage previously correct content, and more steps, stronger static decomposition, or generic reflective feedback do not consistently guarantee structural correctness. Our central claim is that recovering structured diagrams

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

with a fixed raster generator requires *verified and generator-aware decomposition*, grounded in both the target generator and the accepted intermediate state.

Verified decomposition depends on reliable fine-grained checking. To decide whether an intermediate result should be accepted, retried, or further decomposed, the system must judge structural correctness at the level of objects, relations, and unwanted extra content [1, 6, 25]. Our experiments show substantial performance gaps among vision-language models on such structured verification, indicating that verification is a key bottleneck [11, 37, 38].

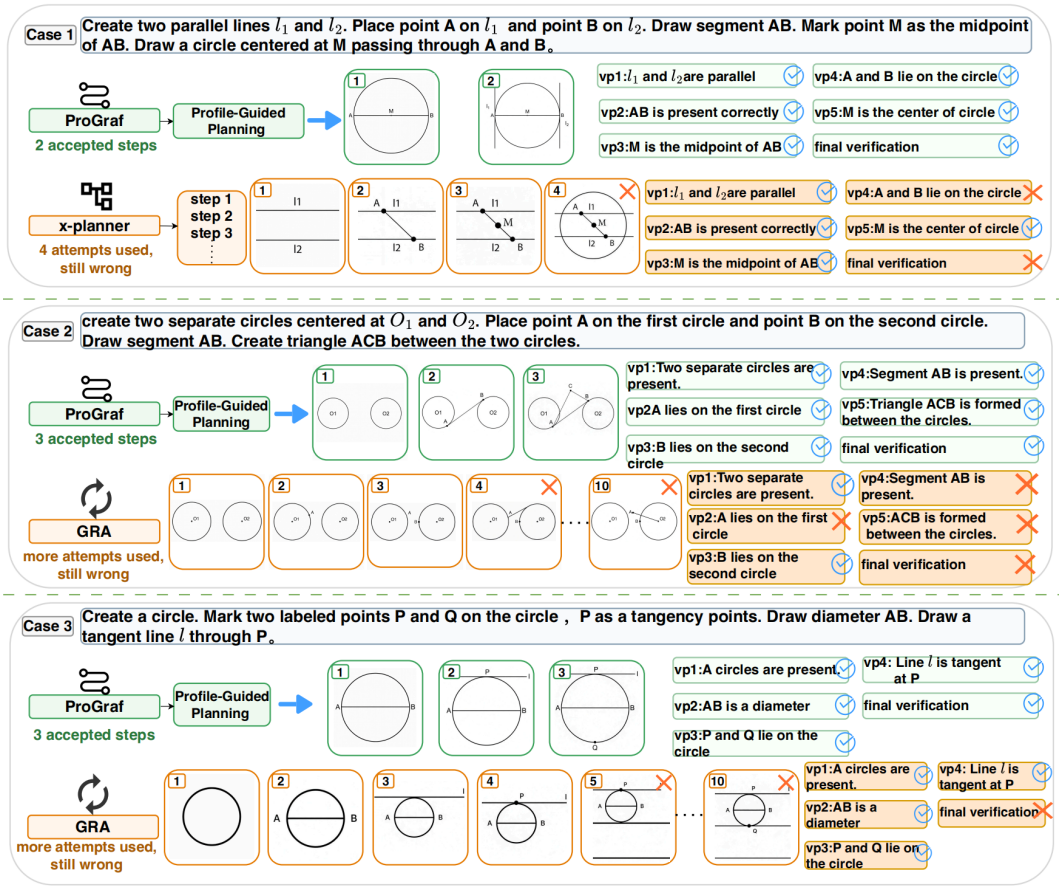


Figure 1: Qualitative comparison of ProGraf and agent baselines on structured generation hard cases. For each case, we show the accepted intermediate states produced by ProGraf, the failed attempt trajectory of a baseline agent, and the corresponding verification outcomes.

Generator awareness is equally important because different image generators exhibit different strengths and weaknesses across categories and operation types. We therefore derive model-specific capability profiles from benchmark annotations and historical generator outputs. As previewed in Figure 1, **ProGraf** uses these profiles to choose lower-risk construction orders, isolate fragile relations into verifiable steps, and decide whether to accept, retry, further decompose, or replan based on VP-grounded structural checks. Operating through the generation and editing APIs of a fixed black-box raster generator, ProGraf directs its budget toward verifier-approved progress on structured diagrams.

To study this setting, we build a benchmark of 304 structured non-natural image tasks spanning geometry, circuit, flowchart, and general diagrams, each paired with fine-grained verification points. Based on this benchmark, ProGraf adaptively decomposes hard tasks into more reliable intermediate generation or editing steps. We compare it with fixed-step baselines and two stronger agent baselines: an X-Planner-inspired static structured-action agent and a Talk2Image-inspired generic reflective closed-loop agent, denoted as **GRA**. Experiments on generator-specific hardest subsets, formed by prompts that fail in all three original trials, show that, under the same black-box raster generator

and call budget, ProGraf consistently outperforms all baselines, while removing benchmark-derived profiling leads to clear performance degradation.

Our contributions are three-fold:

- We introduce a VP-grounded benchmark for structured non-natural image generation across geometry, circuit, flowchart, and general diagrams. Its fine-grained VPs support generator evaluation, verifier evaluation, and benchmark-derived capability profiling.
- We evaluate VLMs as structured diagram verifiers and show substantial model- and category-level gaps, revealing VP-grounded verification as a nontrivial bottleneck for closed-loop structured generation.
- We propose **ProGraf**, a profile-guided closed-loop planner for structured-diagram recovery with black-box raster generators. It uses generator-specific benchmark profiles to guide decomposition, retry, and degradation, while committing only verifier-approved states. On generator-specific hardest subsets, ProGraf consistently improves over fixed-step and adapted agent baselines, and profiling ablations confirm the value of profile alignment.

2 Related Work

Structured Diagram and Non-Natural Image Generation. Recent work has begun to study structured visual generation as a distinct problem from open-ended natural image synthesis [7, 9, 14]. Prior benchmarks and frameworks for text-to-diagram or structured-visual generation emphasize logical organization, editability, and structural correctness rather than purely aesthetic quality [30, 33]. More recent work on structured visuals further highlights that such tasks require adherence to explicit structural rules and are fundamentally different from controllable natural image generation [8, 22, 31, 44]. Unlike prior benchmarks that primarily evaluate final text-image alignment or diagram generation quality, our benchmark is designed to serve two roles: evaluating structured generation and extracting generator-specific profiles for downstream planning. In this way, we study a unified setting that jointly connects fine-grained VP-based evaluation, structured verification, benchmark-derived generator profiling, and adaptive recovery on generator-specific hardest subsets.

Diagram-specific and graphics-program methods offer alternative output interfaces. DiagrammerGPT combines LLM-generated diagram plans with a specialized diagram generator and text-label rendering [39]. AutomaTikZ generates TikZ programs from text [2], while DeTikZify synthesizes TikZ programs from figures and sketches [3]; these programs support editable vector graphics. ProGraf addresses the complementary setting of improving a fixed black-box raster generator through profile-guided planning and verified editing.

Planning, Decomposition, and Agentic Image Generation/Editing. A growing line of work uses multimodal planning, decomposition, or agent-based orchestration to improve image generation and editing [4, 16, 18, 28, 34, 36, 41, 43]. Classical instruction-based editing methods and iterative image-to-image techniques already show the value of structured intermediate control [10, 12, 15, 19, 20, 32, 35]. More recent agentic systems push further toward explicit planning and closed-loop execution.

X-Planner [36] decomposes complex editing instructions into simpler sub-instructions, predicts edit-type-specific anchors and control signals such as masks or bounding boxes, and iteratively executes them with compatible editing models. Talk2Image [18] targets dialogue-based image generation and editing, combining intention parsing over dialogue history, DAG-based multi-agent scheduling, and feedback-driven refinement. These works demonstrate the value of planning and feedback for image generation, but they mainly address natural-image editing or multi-turn creative interaction. In contrast, our setting focuses on benchmarked structured non-natural diagrams, where the planner is controlled by benchmark-derived generator profiles and VP-grounded structural verification. Therefore, we implement adapted baselines inspired by their planning and reflection principles rather than direct reproductions; details are in Appendix A.8.

Vision-Language Models as Verifiers. Vision-language models have been widely evaluated on multimodal understanding and reasoning benchmarks [5, 11, 17, 37, 38], and recent work has also explored stronger structured critiques and verification-oriented training for generation and editing evaluation [1, 6, 13, 25, 29, 40]. However, most prior studies focus on broad visual reasoning or general-purpose judging. Our focus is different: we evaluate VLMs as fine-grained verifiers

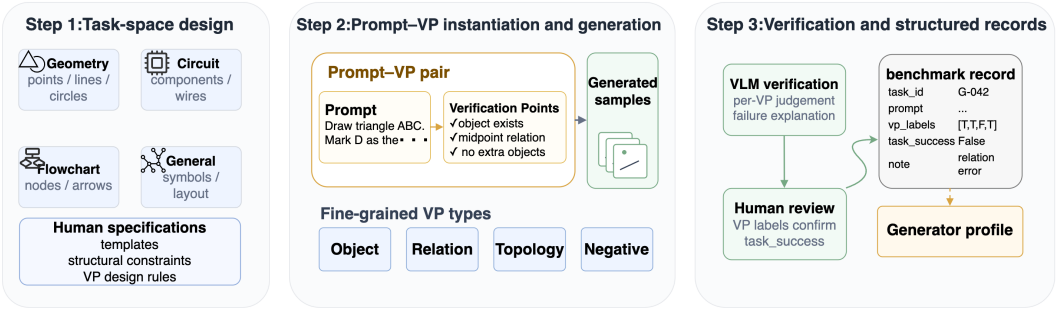


Figure 2: Overview of benchmark construction. We define structured task categories, instantiate prompt–VP pairs, evaluate generated images with VP-level checks, and use the resulting records for generator evaluation, verifier evaluation, and generator-specific profile extraction.

for structured non-natural images, where the model must judge not only object presence but also structural relations and extra erroneous content under a unified VP-based protocol. These verification challenges motivate our joint study of generator capabilities and VP-based checking.

3 Benchmark Design

We construct a benchmark for structured non-natural image generation. It contains 304 tasks across four categories: **Geometry**, **Circuit**, **Flowchart**, and **General Diagrams**. Each task consists of a natural-language prompt and a set of fine-grained **Verification Points (VPs)**. Unlike open-ended natural image generation, these tasks are correctness-sensitive: a visually plausible output can still be wrong if it misses required elements, violates structural relations, or introduces extra content. Figure 2 summarizes the benchmark construction process.

3.1 Task and VP Construction

The benchmark covers structured visual tasks with different constraint types. Geometry tasks involve points, lines, circles, polygons, and relations such as tangency, perpendicularity, and midpoint constraints. Circuit tasks require schematic components and correct electrical topology. Flowchart tasks emphasize node identity, standard symbols, layout, and directed connectivity. General diagrams include abstract graphs, tables, simple layouts, and other structured non-photographic diagrams.

Tasks are created through a human-designed and LLM-assisted process. For each category, we first define a task template, target elements, structural constraints, and desired complexity levels. Candidate prompts are then generated in small batches: in each round, the LLM is given the target category, desired VP count, complexity requirement, and previously accepted prompts, and is instructed to avoid duplicate or near-duplicate tasks. Human reviewers filter and revise the candidates to remove ambiguous, overly similar, or visually underspecified prompts. The LLM is therefore used only for candidate drafting, while the final prompts and VPs are manually checked for clarity, diversity, and verifiability. Full prompt–VP examples are provided in Appendix A.2.

Each task is paired with VPs that decompose correctness into checkable conditions. We group VPs into three types: **positive object constraints**, which check whether required entities are present; **relation and topology constraints**, which check spatial, geometric, or connectivity relations; and **task-specific negative constraints**, which check whether explicitly forbidden structures are absent. The number of VPs reflects the amount of structure that must be verified, so we use VP count as an implicit complexity signal.

3.2 Review Protocol and Hardest Subsets

For each generated image, a VLM first predicts VP-level correctness as preliminary annotation. Two graduate students with computer-science backgrounds independently review the image using the original prompt and VPs, judging each VP and overall task completion without seeing each other's decisions or the VLM verdict. Disagreements are rechecked against the predefined VP criteria,

discussed, and jointly adjudicated. These human-reviewed labels form the final ground truth for both generation and verifier evaluation.

The reviewed records are used for full-benchmark generator evaluation, VLM verifier evaluation, and generator-specific capability profiling. In ProGraf, the runtime verifier is used only for online step acceptance, retry, and replanning; it is not treated as ground truth during final reporting. We distinguish task success from VP-level statistics: task success measures whether the whole diagram satisfies the intended task, while VP-level metrics measure correctness over individual verification points.

For method comparison, we construct a generator-specific hardest subset for each closed-source generator. A task is included in the hardest subset of generator g if all three independent original attempts by g fail the task-level evaluation. These subsets identify prompts that are consistently difficult for a given generator and are used to evaluate hard-case recovery by ProGraf and the baseline methods.

4 Methodology

Our goal is to solve structured non-natural image generation through *verified and generator-aware decomposition*. Given a task prompt τ and a target image generator g, ProGraf first derives a benchmark-based capability profile for g, then performs closed-loop planning, generation or editing, step-level verification, and replanning until the task is completed or the budget is exhausted.

4.1 Benchmark-Derived Generator Profiles

Different image generators exhibit different failure tendencies on structured non-natural images. We therefore construct an offline generator profile as a model-level calibration prior. For a generator g, let $H(g)$ denote task-level records aggregating repeated reviewed outputs for each benchmark prompt, including task categories, structural tags, task-level outcomes, false VP texts, and reviewed failure notes. We extract

$$P_g = \Phi(H(g)) = \{C_g, T_g, F_g\}, \tag{1}$$

where Φ is a deterministic extraction procedure.

The category profile C_g contains category-level statistics, i.e., task success rate, average VP count, and support size. The tag profile T_g contains global tag-level statistics, together with strong and weak tags selected from high-support tags. The failure profile F_g contains global VP-failure patterns, global note-failure patterns, and category-specific failure patterns. Since benchmark categories are not equally sized, global failure patterns are ranked by category-balanced scores rather than raw frequencies. This prevents the largest category from dominating the generator-level failure profile. Details of the extraction algorithm, thresholds, and taxonomies are provided in Appendix A.4.

The planner only receives this compressed aggregate profile and does not receive any per-prompt generated image, VP label, task outcome, or reviewed failure note from the benchmark records. Thus, P_g serves as an aggregate generator-level capability prior.

4.2 Profile-Guided Planner

At each planning round t, ProGraf plans only the next step rather than a fixed full trajectory. The planner is conditioned on

$$s_t = (\tau, I_t, A_t, F_t, b_t, P_g), \tag{2}$$

where τ is the task prompt, I_t is the current committed image, A_t is the accepted step history, F_t is the verifier-derived failure context, b_t is the remaining generation budget, and P_g is the benchmark-derived generator profile. Only verified intermediate images are committed; failed candidates are rejected and do not become future planning states.

In implementation, the LLM planner uses a fixed prompt interface with strict JSON output. The prompt instructs the planner to treat P_g as the primary planning constraint, while using task-independent format examples only as secondary guidance for VP style. The planner then converts profile entries into local construction choices. For example, weaknesses in relations such as tangency or perpendicularity in P_g can guide the planner to isolate them into separate steps. After a failed step,

verifier feedback and profile priors jointly guide whether to retry, further decompose, or replan from the last committed state.

The planner output and controller-selected mode define a structured next-step action

$$a_t = (d_t, m_t, q_t, u_t, V_t, \Delta_t), \tag{3}$$

where d_t is the step/finish decision, m_t is the controller-selected generation/editing mode, q_t is the local subgoal, u_t is the generation or editing instruction, V_t is the step-level VP set used to verify the candidate image for the current step, and Δ_t is the expected local change. When $d_t = \texttt{finish}$, the planner also supplies completion VPs for full-task verification. Since the prompt interface is fixed while P_g, I_t, A_t, and F_t vary, the same task can receive different decompositions depending on the target generator and the verified intermediate state.

4.3 Closed-Loop Execution

Given the next-step action a_t, the generator executes instruction u_t under mode m_t to produce a candidate image $\hat{I}_{t+1}$. A VP-based step verifier checks V_t and expected change Δ_t. A passing candidate is committed as I_{t+1}, with the step added to A_{t+1}. Each step allows up to K_{attempt} attempts, including the initial attempt; retries append the verifier's hint. Exhausting retries adds failure context and triggers replanning from the last committed state within degradation limit $D_{\max}$. At each finish proposal, the planner supplies completion VPs; deterministic rules supplement them with applicable relation checks and a no-extra-content check to form V_t^{final}. A rejected final check feeds its reason and checked VPs back to the planner for repair; subsequent finish proposals may revise the completion VPs. An accepted step resets degradation depth. Budget exhaustion ends execution before another planner round or final check.This accepted-state commitment is central to ProGraf: unverified candidates are never used as future intermediate states. The full algorithm is provided in Appendix A.6.

5 Experiments

5.1 Experimental Setup

We evaluate on our benchmark of 304 structured non-natural image tasks spanning geometry, circuit, flowchart, and general diagrams. Each task is paired with fine-grained verification points (VPs) covering object, relation, and negative constraints. Table 1 reports the benchmark statistics.

Models. For full-benchmark generator evaluation, we test five image generators: three closed-source models, grok-imagine, seedream5-lite, and Nanobanana2, and two open-source models, GLM-Image and FLUX2-9B. Each prompt is sampled independently three times, and these samples are used to measure original generator performance and construct generator-specific capability profiles. For verifier evaluation, we test five VLMs: Qwen3-VL-8B-Instruct, Qwen3-VL-32B, InternVL3-8B, Gemini-3.1-Flash-Lite, and Gemini-3.1-Pro-Preview. In ProGraf, both the planner and the VLM-based verifier are instantiated with Gemini-3.1-Pro-Preview.

Evaluation protocol. Generated images are first evaluated at the VP level by a VLM-based verifier and then reviewed manually for the final statistics. Human review checks both VP-level correctness and task-level correctness. This reviewed result is used for generator evaluation, verifier evaluation, and hardest-subset construction. All experiments use a square canvas.

Hardest subsets. For method comparison, we construct a generator-specific hardest subset for each closed-source generator. A prompt is included in the hardest subset of generator g if all three original independent attempts by g fail the task-level evaluation. We focus the hardest-subset comparison on the three closed-source generators; the two open-source generators have very low full-benchmark success rates and struggle with basic diagram construction. This comparison evaluates hard-case recovery through step ordering, verification, and replanning.

Baselines and planner settings. We compare ProGraf with two types of baselines. The first type consists of fixed-step static-decomposition controls with $K{=}3$ and $K{=}5$. These baselines

Table 1: Benchmark statistics. We report the number of tasks, total VPs, average and median VPs per task, the distribution of VP types, and the average prompt length.

Category	Tasks	VPs	Avg. VP	Med. VP	Pos. VP	Rel. VP	Neg. VP	Avg. Prompt Tokens
Geometry	154	547	3.55	3	41.13%	44.61%	14.26%	38.64
Circuit	50	400	8.00	8	71.50%	21.75%	6.75%	71.18
Flowchart	50	408	8.16	8	85.29%	13.48%	1.23%	66.06
General	50	400	8.00	8	38.75%	41.25%	20.00%	76.62
Overall	304	1755	5.77	6	57.78%	31.40%	10.83%	54.75

generate a one-shot K-step plan, execute exactly K generation/editing calls, accept each intermediate output without step-level rollback, and perform only final verification. They test whether static multi-step decomposition alone is sufficient. The second type consists of budget-matched adapted agent baselines: an X-Planner-inspired static structured-action baseline and a Talk2Image-inspired generic reflective agent baseline (GRA). Together with ProGraf, these methods are run under the same maximum global budget B=10. The adapted agent baselines can spend budget on retry or feedback-driven execution, but they do not use benchmark-derived generator profiles. ProGraf uses K_{attempt}=3 and $D_{\max}$=2. Details are provided in Appendix A.8.

Metrics. For full-benchmark generator evaluation, we report task success rate, VP-success rate, and VP-level accuracy. Task success rate measures final task-level correctness, VP-success rate measures the fraction of samples for which all benchmark VPs are satisfied, and VP-level accuracy measures correctness over individual benchmark VP instances. For VLM evaluation, we report VP-level accuracy, precision, recall, specificity/TNR, and balanced accuracy, using balanced accuracy as the primary verifier metric because VP labels are imbalanced. For hardest-subset method comparison, we use task success rate (TSR) as the primary metric and report both percentage and count. Runtime VPs are dynamically generated by each method and treated as internal control signals rather than pooled final metrics. We also report average total attempts (ATT), committed-step ratio (CSR), failed attempts per task (FAT), and final verification rejection rate (FVR) to analyze efficiency and recovery behavior. Detailed definitions are provided in Appendix A.5.

Table 2: Full-benchmark evaluation of generators and VLM-based verifiers.

(a) Generator performance on the full benchmark.

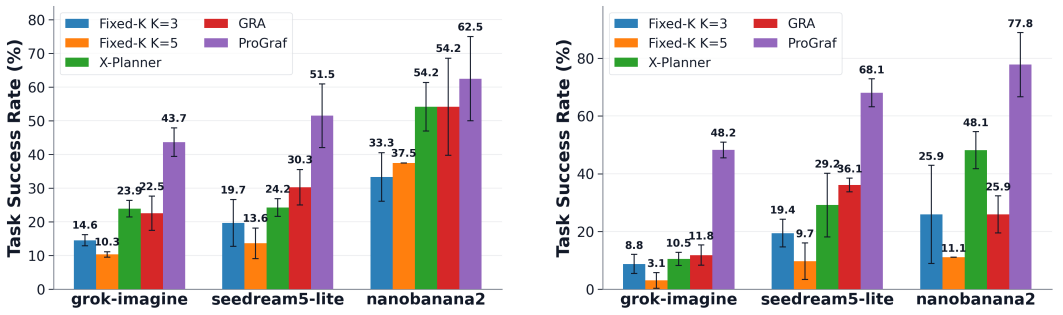
Model	Samples	Task Success	VP-Success	VP-Level Acc.
Nanobanana2	912	80.3	85.7	95.7
seedream5-lite	912	67.3	70.6	90.3
grok-imagine	912	35.6	40.1	73.8
GLM-Image	912	13.8	19.0	52.8
FLUX2-9B	912	5.8	8.2	38.2

(b) VLM performance as structured verifiers.

VLM	Acc.	Prec.	Recall	TNR	Balanced Acc.
Qwen3-VL-8B-Instruct	75.37	93.54	78.77	34.29	56.53
Qwen3-VL-32B	81.08	93.79	85.15	31.93	58.54
InternVL3-8B	54.79	94.57	54.16	62.42	58.29
Gemini-3.1-Flash-Lite	75.03	95.94	76.19	61.04	68.61
Gemini-3.1-Pro-Preview	82.65	97.13	83.69	70.10	76.89

5.2 Full-Benchmark Evaluation

Table 2 reports the full-benchmark results for both image generators and VLM-based verifiers. For generation, Nanobanana2 performs best overall, reaching 80.3% task success, 85.7% VP-success, and 95.7% VP-level accuracy, followed by seedream5-lite. grok-imagine is substantially weaker, while the two open-source models perform poorly on this benchmark. The category-wise results show similar trends: flowchart is the easiest category, whereas geometry remains challenging, especially for



(a) Geometry hardest subset.

(b) Other hardest subset, including circuit, flowchart, and general diagrams.

Figure 3: Main results on generator-specific hardest subsets. ProGraf is compared with fixed-step baselines and adapted agent baselines on geometry and non-geometry subsets. Bars show mean task success rate (TSR). Error bars denote standard deviation over three independent runs.

weaker generators. These results indicate clear model- and category-specific capability differences, motivating the use of generator-specific profiles.

For verification, Gemini-3.1-Pro-Preview achieves the best overall Balanced Accuracy of 76.89%. Qwen3-VL-32B obtains high Recall but much lower Specificity/TNR, suggesting that recognizing satisfied constraints is easier than detecting violated ones. Since VP labels are imbalanced, we emphasize Balanced Accuracy over Accuracy alone. Overall, the full-benchmark evaluation shows that both generation and verification remain nontrivial in structured non-natural image tasks, motivating our profile-guided and VP-grounded planning framework.

5.3 Main Results on Hardest Subsets

We evaluate ProGraf on the generator-specific hardest subsets defined in Section 5.1, measuring recovery of tasks that failed all three original trials.

Figure 3 reports task success rate on geometry and non-geometry hardest subsets. ProGraf achieves the highest success rate for all three generators on both subsets, with particularly large gains on grok-imagine and seedream5-lite. On geometry, ProGraf improves TSR to 43.7%, 51.5%, and 62.5% for grok-imagine, seedream5-lite, and Nanobanana2, respectively. On non-geometry tasks, the corresponding rates are 48.2%, 68.1%, and 77.8%. The improvement is substantial over fixed-step baselines and remains clear against stronger adapted agent baselines, including the X-Planner-inspired static agent and the generic reflective agent (GRA). Increasing the number of fixed steps from $K{=}3$ to $K{=}5$ does not reliably improve performance, suggesting that static decomposition alone is insufficient and can even amplify editing errors.

Figure 4 further reports metric-level differences between ProGraf and representative baselines. ProGraf consistently improves task success rate, while also reducing final-verification rejections in most comparisons. The gains are largest against fixed-step decomposition, but remain positive against adapted agent baselines on task-level recovery. Process metrics show a more nuanced pattern: ProGraf often improves committed-step ratio and attempt efficiency over GRA, but comparisons against the X-Planner-inspired baseline show mixed changes in committed-step ratio and failed attempts per task. This indicates that higher task success does not simply come from accepting more intermediate edits; rather, the accepted edits must preserve the global structure and move the diagram toward a verifiable final state. For Nanobanana2, the hardest subsets are smaller because the base generator is stronger, so we report raw counts in addition to percentages and treat these results as supportive rather than the primary evidence.

Profiling and efficiency. Finally, we verify that the gains are not explained by retry and replanning alone. On the geometry hardest subset, removing generator-specific profiling reduces TSR from 51.52% to 30.30% for seedream5-lite and from 62.50% to 54.17% for Nanobanana2. We further conduct a shuffled-profile ablation on the grok-imagine geometry hardest subset, where the planner keeps the same profile format, verifier, and budget, but replaces the correct grok-imagine profile with

	TSR Δ↑	ATT Red.↑	CSR Δ↑	FAT Red.↑	FVR Red. ↑
grok-imagine | vs K=3	+29.1pp	-3.5	N/A	N/A	+80.6pp
grok-imagine | vs X-Planner	+19.7pp	+0.4	-5.1pp	-0.2	+17.0pp
grok-imagine | vs GRA	+21.1pp	+1.7	+31.4pp	-0.5	+50.6pp
seedream5-lite | vs K=3	+31.8pp	-1.6	N/A	N/A	+50.9pp
seedream5-lite | vs X-Planner	+27.3pp	+0.6	+0.2pp	+0.1	+5.4pp
seedream5-lite | vs GRA	+21.2pp	+1.4	+29.7pp	-0.5	+7.9pp
nanobanana2 | vs K=3	+29.2pp 15/24 | 8/24	-0.4	N/A	N/A	+38.2pp 1/16 | 8/18
nanobanana2 | vs X-Planner	+8.3pp 15/24 | 13/24	+1.5	+2.0pp	+1.0	+3.8pp 1/16 | 2/20
nanobanana2 | vs GRA	+8.3pp 15/24 | 13/24	+1.8	+43.9pp	+1.0	+10.4pp 1/16 | 4/24

(a) Geometry.

	TSR Δ↑	ATT Red.↑	CSR Δ↑	FAT Red.↑	FVR Red. ↑
grok-imagine | vs K=3	+39.5pp	-3.8	N/A	N/A	+84.3pp
grok-imagine | vs X-Planner	+37.7pp	-2.8	+11.1pp	-2.7	+46.1pp
grok-imagine | vs GRA	+36.4pp	+2.6	+29.8pp	-0.2	+81.3pp
seedream5-lite | vs K=3	+48.6pp	-1.3	N/A	N/A	+78.5pp
seedream5-lite | vs X-Planner	+38.9pp	+0.3	+0.5pp	-1.5	+32.1pp
seedream5-lite | vs GRA	+32.0pp	+4.3	+37.7pp	0.0	+61.3pp
nanobanana2 | vs K=3	+51.9pp 21/27 | 7/27	-1.5	N/A	N/A	+69.5pp 1/22 | 20/27
nanobanana2 | vs X-Planner	+29.6pp 21/27 | 13/27	+1.0	-5.4pp	-0.4	+33.5pp 1/22 | 8/21
nanobanana2 | vs GRA	+51.9pp 21/27 | 7/27	+2.8	+53.6pp	-0.1	+69.5pp 1/22 | 20/27

(b) Non-geometry.

Figure 4: Differences between full-profile ProGraf and the corresponding baseline. TSR and CSR show increases; ATT, FAT, and FVR show reductions. CSR is the committed-step ratio; FAT is failed attempts per task. Fixed-K baselines have N/A for CSR and FAT. Controller metrics are interpreted alongside human-reviewed TSR. Nanobanana2 cells also show pooled counts as ProGraf | baseline.

a profile extracted from another generator. The shuffled-profile variant achieves 35.2% TSR with 6.94 average attempts per prompt, lower than ProGraf with the correct profile, while using more attempts. This suggests that mismatched profiles can still provide generic structured-diagram guidance, since different generators share some common failure modes, but correct generator-profile alignment yields additional gains and more efficient recovery. ProGraf is allowed a larger maximum adaptive attempt budget than fixed-step baselines, but this budget is allocated through verification-guided retry and replanning rather than uniformly longer trajectories. Additional analyses of profile alignment, leave-category-out profile transfer, profiling components, budget sensitivity, accuracy–efficiency trade-offs, and qualitative limitations are provided in Appendix B.

5.4 Held-Out Profiles and Budget-Matched Controls

We additionally evaluate seedream5-lite Geometry using five-fold cross-fitting over all 154 prompts. Folds use prompt text only and keep identified high-overlap prompt clusters together. Before extracting each profile, we exclude all historical outputs and annotations from the evaluation fold; the other four Geometry folds and non-Geometry calibration records are retained. The held-out folds cover the same 22 hardest prompts once per run. Across three runs, human-reviewed TSR is $48.48 \pm 2.62\%$ (sample SD), compared with 51.52% under the original full-profile setting, with ATT of 5.58. Recovery therefore persists after removing the evaluated prompts and their identified high-overlap clusters from profile construction.

Table 3: Held-out seedream5-lite Geometry controls with $B = 10$. Full reports the mean $\pm$ sample SD of three runs; each other condition is run once. TSR uses human-reviewed final outputs.

Condition	TSR (%)	ATT
Full ProGraf	48.48 ± 2.62	5.58
Best-of-10	31.82	10.00
w/o VP grounding	31.82	4.77
w/o rollback	18.18	7.14
Retry-only	27.27	5.68
Profile-only	27.27	4.36

Best-of-10 spends all ten calls on independent complete images selected by the same verifier; Full attains a higher mean TSR with fewer calls on average. The component controls isolate VP feedback, accepted-state rollback, adaptive replanning, and profile-guided planning alone; the three-run w/o-profile ablation is in Appendix B, and Appendix C gives definitions, split details, prompt-level uncertainty, and a concrete profile/planner example.

Verifier error impact. We audit all 34 human-incorrect trajectories from the three held-out Full runs. Nineteen contain an accepted intermediate error that persists into the final output; two contain a usable intermediate state that is falsely rejected, followed by budget exhaustion. Review identifies no contributing verifier error in the remaining 13 budget-exhausted trajectories. These observations show how false acceptance preserves incorrect structure and false rejection consumes repair budget. Midpoint and perpendicular cases are examined in Appendix C.5.

6 Conclusion

We present a VP-grounded benchmark of 304 structured non-natural image tasks and ProGraf, a profile-guided closed-loop planner for recovering failed tasks with a fixed black-box raster generator. Generators vary substantially across categories, and VLM-based structured verifiers show clear gaps, indicating that capability profiling and verification are both central to this setting. On generator-specific hardest subsets, ProGraf consistently outperforms fixed-step and adapted agent baselines; under a prompt-cluster-disjoint held-out split, recovery remains close to the full-profile result, and budget-matched controls suggest that the gain is not explained by additional sampling alone.

Scope and limitations. ProGraf is a recovery policy for raster generators and does not claim superiority over symbolic, SVG, or TikZ-based synthesis, which offer complementary editability and precision. Its profiles are extracted from reviewed calibration records, so transfer to a new generator or domain requires analogous records and verifiable constraints. The clean held-out evidence covers one generator, one category, and 22 hard prompts, most new controls use a single run, and hardest-subset results measure conditional recovery rather than average-case performance. ProGraf remains bounded by the generator's primitive capabilities and by verifier errors that can propagate through the loop (Appendix C.5).

Acknowledgments and Disclosure of Funding

Lan Zhang is the corresponding author. This work was supported by the Shandong Provincial Key R&D Program (Major Technological Innovation Project) "Research and Application of Multi-Agent Collaborative Technologies for Public Safety" (Project No. 2026CXGC010110), China National Natural Science Foundation with No. 62441228, Science and Technology Tackling Program of Anhui Province, No. 202423k09020016. The authors declare no competing interests.

References

[1] Lorenzo Baraldi, Davide Bucciarelli, Federico Betti, Marcella Cornia, Nicu Sebe, and Rita Cucchiara. What changed? detecting and evaluating instruction-guided image edits with multimodal large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16217–16226, 2025.

[2] Jonas Belouadi, Anne Lauscher, and Steffen Eger. Automatikz: Text-guided synthesis of scientific vector graphics with tikz. In *International Conference on Learning Representations*, volume 2024, pages 55917–55943, 2024.

[3] Jonas Belouadi, Simone P Ponzetto, and Steffen Eger. Detikzify: Synthesizing graphics programs for scientific figures and sketches with tikz. *Advances in Neural Information Processing Systems*, 37:85074–85108, 2024.

[4] Tim Brooks, Aleksander Holynski, and Alexei A Efros. Instructpix2pix: Learning to follow image editing instructions. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 18392–18402, 2023.

[5] Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, et al. Are we on the right way for evaluating large vision-language models? *Advances in Neural Information Processing Systems*, 37:27056–27087, 2024.

[6] Sherry X Chen, Yi Wei, Luowei Zhou, and Suren Kumar. Adiee: Automatic dataset creation and scorer for instruction-guided image editing evaluation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 18345–18356, 2025.

[7] Xueqing Deng, Linjie Yang, Qihang Yu, Chenglin Yang, and Liang-Chieh Chen. Leveraging panoptic scene graph for evaluating fine-grained text-to-image generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15107–15116, 2025.

[8] Jiaxin Ge, Zora Zhiruo Wang, Xuhui Zhou, Yi-Hao Peng, Sanjay Subramanian, Qinyue Tan, Maarten Sap, Alane Suhr, Daniel Fried, Graham Neubig, et al. Autopresent: Designing structured visuals from scratch. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 2902–2911, 2025.

[9] Dhruba Ghosh, Hannaneh Hajishirzi, and Ludwig Schmidt. Geneval: An object-focused framework for evaluating text-to-image alignment. *Advances in Neural Information Processing Systems*, 36:52132–52152, 2023.

[10] Yiming Gong, Zhen Zhu, and Minjia Zhang. Instantedit: Text-guided few-step image editing with piecewise rectified flow. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16808–16817, 2025.

[11] Tianrui Guan, Fuxiao Liu, Xiyang Wu, Ruiqi Xian, Zongxia Li, Xiaoyu Liu, Xijun Wang, Lichang Chen, Furong Huang, Yaser Yacoob, et al. Hallusionbench: an advanced diagnostic suite for entangled language hallucination and visual illusion in large vision-language models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14375–14385, 2024.

[12] Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. Prompt-to-prompt image editing with cross attention control. *arXiv preprint arXiv:2208.01626*, 2022.

[13] Kaiyuan Hou, Minghui Zhao, Lilin Xu, Yuang Fan, and Xiaofan Jiang. Tdbench: Benchmarking vision-language models in understanding top-down images. *arXiv e-prints*, pages arXiv–2504, 2025.

[14] Kaiyi Huang, Kaiyue Sun, Enze Xie, Zhenguo Li, and Xihui Liu. T2i-compbench: A comprehensive benchmark for open-world compositional text-to-image generation. *Advances in Neural Information Processing Systems*, 36:78723–78747, 2023.

[15] Shijie Huang, Yiren Song, Yuxuan Zhang, Hailong Guo, Xueyin Wang, and Jiaming Liu. Arteditor: Learning customized instructional image editor from few-shot examples. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17651–17662, 2025.

[16] Yuzhou Huang, Liangbin Xie, Xintao Wang, Ziyang Yuan, Xiaodong Cun, Yixiao Ge, Jiantao Zhou, Chao Dong, Rui Huang, Ruimao Zhang, et al. Smartedit: Exploring complex instruction-based image editing with multimodal large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8362–8371, 2024.

[17] Bohao Li, Rui Wang, Guangzhi Wang, Yuying Ge, Yixiao Ge, and Ying Shan. Seed-bench: Benchmarking multimodal llms with generative comprehension. *arXiv preprint arXiv:2307.16125*, 2023.

[18] Shichao Ma, Yunhe Guo, Jiahao Su, Qihe Huang, Zhengyang Zhou, and Yang Wang. Talk2image: A multi-agent system for multi-turn image generation and editing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 32437–32445, 2026.

[19] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. Sdedit: Guided image synthesis and editing with stochastic differential equations. *arXiv preprint arXiv:2108.01073*, 2021.

[20] Ron Mokady, Amir Hertz, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. Null-text inversion for editing real images using guided diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6038–6047, 2023.

[21] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.

[22] Yuyang Peng, Shishi Xiao, Keming Wu, Qisheng Liao, Bohan Chen, Kevin Lin, Danqing Huang, Ji Li, and Yuhui Yuan. Bizgen: Advancing article-level visual text rendering for infographics generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 23615–23624, 2025.

[23] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International conference on machine learning*, pages 8821–8831. Pmlr, 2021.

[24] 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*, pages 10684–10695, 2022.

[25] Suho Ryu, Kihyun Kim, Eugene Baek, Dongsoo Shin, and Joonseok Lee. Towards scalable human-aligned benchmark for text-guided image editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18292–18301, 2025.

[26] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in neural information processing systems*, 35:36479–36494, 2022.

[27] Guibao Shen, Luozhou Wang, Jiantao Lin, Wenhang Ge, Chaozhe Zhang, Xin Tao, Di Zhang, Pengfei Wan, Guangyong Chen, Yijun Li, et al. Scene graph guided generation: Enable accurate relations generation in text-to-image models via textural rectification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15437–15446, 2025.

[28] Shelly Sheynin, Adam Polyak, Uriel Singer, Yuval Kirstain, Amit Zohar, Oron Ashual, Devi Parikh, and Yaniv Taigman. Emu edit: Precise image editing via recognition and generation tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8871–8879, 2024.

[29] Zhi Rui Tam, Ya-Ting Pai, Yen-Wei Lee, and Yun-Nung Chen. Vistw: Benchmarking vision-language models for traditional chinese in taiwan. *arXiv preprint arXiv:2503.10427*, 2025.

[30] 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. *arXiv preprint arXiv:2508.01237*, 2025.

[31] Yinghao Tang, Xueding Liu, Boyuan Zhang, Tingfeng Lan, Yupeng Xie, Jiale Lao, Yiyao Wang, Haoxuan Li, Tingting Gao, Bo Pan, et al. Igenbench: Benchmarking the reliability of text-to-infographic generation. *arXiv preprint arXiv:2601.04498*, 2026.

[32] Narek Tumanyan, Michal Geyer, Shai Bagon, and Tali Dekel. Plug-and-play diffusion features for text-driven image-to-image translation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1921–1930, 2023.

[33] 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 Computer Vision and Pattern Recognition Conference*, pages 13315–13325, 2025.

[34] Mingde Yao, Zhiyuan You, Tam-King Man, Menglu Wang, and Tianfan Xue. Photoagent: Agentic photo editing with exploratory visual aesthetic planning. *arXiv preprint arXiv:2602.22809*, 2026.

[35] Ruijie Ye, Jiayi Zhang, Zhuoxin Liu, Zihao Zhu, Siyuan Yang, Li Li, Tianfu Fu, Franck Dernoncourt, Yue Zhao, Jiacheng Zhu, et al. Agent banana: High-fidelity image editing with agentic thinking and tooling. *arXiv preprint arXiv:2602.09084*, 2026.

[36] Chun-Hsiao Yeh, Yilin Wang, Nanxuan Zhao, Richard Zhang, Yuheng Li, Yi Ma, and Krishna Kumar Singh. Beyond simple edits: X-planner for complex instruction-based image editing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 11991–11999, 2026.

[37] Weihao Yu, Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Zicheng Liu, Xinchao Wang, and Lijuan Wang. Mm-vet: Evaluating large multimodal models for integrated capabilities. *arXiv preprint arXiv:2308.02490*, 2023.

[38] Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9556–9567, 2024.

[39] Abhay Zala, Han Lin, Jaemin Cho, and Mohit Bansal. Diagrammergpt: Generating open-domain, open-platform diagrams via llm planning. *arXiv preprint arXiv:2310.12128*, 2023.

[40] Jipeng Zhang, Kehao Miao, Renjie Pi, Zhaowei Wang, Runtao Liu, Rui Pan, and Tong Zhang. Vl-genrm: Enhancing vision-language verification via vision experts and iterative training. *arXiv preprint arXiv:2506.13888*, 2025.

[41] Lei Zhang, Junjiao Tian, Zhipeng Fan, Kunpeng Li, Jialiang Wang, Weifeng Chen, Markos Georgopoulos, Felix Juefei-Xu, Yuxiang Bao, Julian McAuley, et al. Think in strokes, not pixels: Process-driven image generation via interleaved reasoning. *arXiv preprint arXiv:2604.04746*, 2026.

[42] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3836–3847, 2023.

[43] Sashuai Zhou, Qiang Zhou, Jijin Hu, Hanqing Yang, Yue Cao, Junpeng Ma, Yinchao Ma, Jun Song, Tiezheng Ge, Cheng Yu, et al. Unified thinker: A general reasoning modular core for image generation. *arXiv preprint arXiv:2601.03127*, 2026.

[44] Le Zhuo, Songhao Han, Yuandong Pu, Boxiang Qiu, Sayak Paul, Yue Liao, Yihao Liu, Jie Shao, Xi Chen, Si Liu, et al. Factuality matters: When image generation and editing meet structured visuals. *arXiv preprint arXiv:2510.05091*, 2025.

A Additional Experimental Details

A.1 Category-wise Generator Performance

Table 4 reports category-wise generator performance on the full benchmark. These results complement the overall full-benchmark results in the main paper and show clear category-specific differences across generators.

Table 4: Category-wise generator performance on the full benchmark. VP Acc. denotes VP-level accuracy averaged over individual verification points. All values are percentages.

Model	Geometry		Circuit		Flowchart		General	
	TSR	VP Acc.	TSR	VP Acc.	TSR	VP Acc.	TSR	VP Acc.
Nanobanana2	75.8	92.3	90.0	97.2	95.3	99.4	69.3	94.6
seedream5-lite	65.8	86.2	68.7	91.5	76.7	97.4	61.3	85.8
grok-imagine	38.5	61.8	32.7	74.6	39.3	92.6	26.0	67.7
GLM-Image	13.4	31.7	2.0	44.5	33.3	88.4	7.3	51.1
FLUX2-9B	6.1	19.1	1.3	34.4	9.3	64.8	6.0	40.0

A.2 Benchmark Examples

Table 5 provides representative full prompt–VP pairs from the four benchmark categories. These examples complement the concise task descriptions in Section 3 and illustrate how each task is decomposed into checkable structural conditions.

A.3 Generator-specific Hardest Subsets

Table 6 reports the size of generator-specific hardest subsets. A prompt is included if the corresponding generator fails in all three original full-benchmark trials. Stronger generators naturally have fewer all-fail prompts.

A.4 Profile Extraction Details

We provide the details of the profile extraction function Φ used in Section 4.1. The extraction procedure is deterministic and uses fixed support thresholds, ranking rules, and predefined failure taxonomies. For each generator g, it takes reviewed benchmark records $H(g)$ as input and outputs a generator profile $P_g = \{C_g, T_g, F_g\}$. The category profile C_g contains category-level statistics. The tag profile T_g contains tag-level statistics, strong tags, and weak tags. The failure profile F_g contains global VP-failure patterns, global note-failure patterns, and category-specific failure patterns.

The benchmark records $H(g)$ are used only to summarize aggregate generator tendencies. They are not exposed to the planner as prompt-specific examples, generated images, reviewed notes, or task outcomes. In addition, the fixed format demonstrations used by the planner are not used during profile extraction. The taxonomy matching step is rule-based and deterministic: false VP texts are mapped to predefined VP-failure types, and reviewed notes are mapped to predefined note-failure types.

Category and tag statistics. For each category c, we collect the subset $H_g(c) = \{h \in H(g) : \mathrm{category}(h) = c\}$ and store its task success rate, average VP count, and support size in $C_g.\mathrm{category_stats}$. For each structural tag z, we similarly collect $H_g(z) = \{h \in H(g) : z \in$

Table 5: Representative full benchmark prompts and verification points.

Category	Prompt example	Verification Points
Geometry	On a pure white background, create triangle ABC. Draw a circle passing through A, B, and C. Draw a tangent line l through A to that circle. Draw segment OA from the circle center O to point A.	Triangle ABC is formed; A, B, and C lie on the circle; line l is tangent to the circle at A; segment OA is a radius to A; no unrelated objects are added.
Circuit	Draw a 5V linear regulator circuit. Show a DC input source on the left, a three-terminal regulator U1 in the middle, and an output node labeled +5V on the right. Add capacitor C1 from the regulator input to ground, and capacitor C2 from the regulator output to ground. Connect the regulator ground pin to ground. Use a standard clean schematic style.	Input source is present; regulator U1 is present; +5V output is present; capacitor C1 is connected from input to ground; capacitor C2 is connected from output to ground; regulator ground is connected to ground; topology is source to regulator to output; no extra load is added.
Flowchart	Draw a monochrome flowchart showing a simple order process: Start, Receive Order, Pack Item, Ship Item, End. Use standard flowchart symbols and a clean vertical layout. Connect each step with a downward arrow.	Start node is present; Receive Order node is present; Pack Item node is present; Ship Item node is present; End node is present; the steps are connected by downward arrows; no branch is added.
General	Draw a monochrome abstract graph with six nodes labeled A, B, C, D, E, and F arranged in two offset rows. The top row is A-B-C and the bottom row is D-E-F. Connect A-B, B-C, D-E, and E-F. Also connect B-E, A-D, and C-F. Add one extra diagonal A-E.	All six nodes are present; top and bottom row edges are present; vertical-like edges A-D, B-E, and C-F are present; diagonal A-E is present; no diagonal B-D is present; no diagonal B-F is present; two-row structure is clear; no extra edge is added.

Table 6: Generator-specific hardest subset sizes.

Generator	Geometry	Other	Circuit	Flowchart	General
grok-imagine	71	76	24	22	30
seedream5-lite	22	24	5	3	16
Nanobanana2	8	9	1	0	8

$\mathrm{tags}(h)\}$ and store its task success rate, average VP count, and support size in $T_g.\mathrm{tag_stats}$. Support counts distinct benchmark tasks. Success rates are computed by first averaging human-reviewed binary task outcomes over repeated outputs for each task, then averaging these per-task rates within the category or tag. Tags supported by fewer than $n_{\min} = 3$ distinct tasks are excluded from the strong/weak tag summary. Among the remaining tags, we retain the top $K = 8$ tags with the highest task success rates as $T_g.\mathrm{strong_tags}$, and the top $K = 8$ tags with the lowest task success rates as $T_g.\mathrm{weak_tags}$. Ties are broken by larger support.

Category-balanced failure scoring. Because the benchmark categories are not equally sized, raw failure counts can be dominated by the largest category. We therefore rank global failure patterns using a category-balanced score. For a failure pattern r, we define

$$\mathrm{Score}_g(r) = \frac{1}{|\mathcal{C}_r|} \sum_{c \in \mathcal{C}_r} \frac{\mathrm{count}_{g,c}(r)}{|H_g(c)|}, \tag{4}$$

where $\mathcal{C}_r$ is the set of categories in which pattern r appears, $\mathrm{count}_{g,c}(r)$ counts matched occurrences of r across repeated reviewed outputs for generator g in category c, whereas $|H_g(c)|$ counts distinct benchmark tasks in that category. We retain up to 10 VP-failure patterns with total frequency at least 3 and up to 10 note-failure patterns with total frequency at least 2, ranked by this score with total frequency as the tie-breaker.

Category-specific failure patterns. In addition to the global failure profile, we store category-specific failure patterns for each category c. These patterns are extracted by applying the same

taxonomy mapping to false VP texts and reviewed notes within $H_g(c)$, and are ranked by their within-category frequency. They are used when the planner receives the target task category, allowing the same generator profile to express both global tendencies and category-dependent failure modes.

A.5 Metric Definitions

For hardest-subset experiments, repeated runs are counted as separate task instances. Let N be the number of evaluated task instances. For example, a hardest subset with n prompts and three repeated runs has $N = 3n$ task instances.

Task Success Rate (TSR) is the fraction of task instances whose returned final image satisfies the task under human review:

$$\mathrm{TSR} = \frac{1}{N} \sum_{i=1}^{N} y_i,$$

where $y_i = 1$ for a human-correct image and $y_i = 0$ otherwise, including instances with no returned image. Runtime verifier decisions control execution; human-reviewed labels determine TSR. Run-level error bars use sample SD over repeated runs of the same prompts.

Average Total Attempts (ATT) is the average number of generated/edited candidates per task instance:

$$\mathrm{ATT} = \frac{\sum_{i=1}^{N} a_i}{N},$$

where a_i counts candidates produced for task instance i, including rejected candidates but excluding service errors that produce no candidate. For fixed-K baselines, ATT is fixed by design and equals K.

Committed-Step Ratio (CSR) measures how many proposed steps become committed progress. Let c_i and s_i be the committed and proposed step counts for task instance i, extracted from each method's step or round trace. We define

$$\mathrm{CSR} = \frac{\sum_{i=1}^{N} c_i}{\sum_{i=1}^{N} s_i}.$$

Retries within a proposed step do not add steps to the denominator. A committed step is accepted by the method's runtime verifier or critic, rather than necessarily correct under human review.

Failed Attempts per Task (FAT) uses the recorded controller attempt count e_i and committed-step count c_i:

$$\mathrm{FAT} = \frac{1}{N} \sum_{i=1}^{N} \max(e_i - c_i, 0).$$

It counts recorded attempts that did not become committed steps per task instance and can exceed one. The historical controller counters include recorded runtime/API failures, whereas ATT counts produced candidates only. CSR and FAT correspond to the historical CSV fields named `Step Acceptance Rate` and `Retry Rate`, respectively.

Final Verification Rejection Rate (FVR) uses the last stored final verdict in each task instance. Within a repeat, let $\mathcal{R}$ denote instances with a stored final verdict unmodified by manual overrides and $y_i^{\mathrm{final}} \in \{0, 1\}$ denote that verdict's acceptance decision:

$$\mathrm{FVR} = \frac{|\{i \in \mathcal{R} : y_i^{\mathrm{final}} = 0\}|}{|\mathcal{R}|}.$$

Runs without such a verdict are excluded from this denominator but remain counted in TSR. A task rejected and subsequently accepted contributes its last decision. Reported FVR averages the per-repeat rates with nonempty denominators. Nanobanana2 heatmap cells instead use pooled rejected/verified counts across repeats, matching the displayed counts.

Fixed-K baselines have N/A for CSR and FAT because they lack the step-level accept/reject controller being compared. TSR measures human-reviewed correctness and ATT measures generation cost. CSR, FAT, and FVR describe controller behavior and are interpreted alongside TSR; accepting incorrect states can raise CSR and lower FVR.

A.6 ProGraf Algorithm

Algorithm 1 ProGraf: Profile-Guided Closed-Loop Planning

Require: Task prompt τ, generator g, profile P_g, budget B, attempt limit K_{attempt}, degradation limit $D_{\max}$
Ensure: (I, status): last committed image (possibly $\emptyset$) and system status

Initialize committed image $I \leftarrow \emptyset$, accepted history $A \leftarrow \emptyset$, execution-failure context $F \leftarrow \emptyset$
Initialize budget $b \leftarrow B$, depth $\delta \leftarrow 0$, planning round $t \leftarrow 0$
while $b > 0$ and $t < 3B$ **do**
  $t \leftarrow t + 1$
  Form planner state $s = (\tau, I, A, F, b, P_g)$
  Form $a = (d, m, q, u, V, \Delta)$ from planner output and controller-selected m, with $d \in \{\texttt{step}, \texttt{finish}\}$
  if $d = \texttt{finish}$ **then**
    Form V_t^{final} from this proposal's completion VPs and deterministic guard checks
    $(p, \psi) \leftarrow \text{FinalVerify}(I, V_t^{\text{final}})$
    if $p = \text{true}$ **then**
      return $(I, \texttt{completed})$
    else
      Set $F \leftarrow \text{FinalFail}(\psi, V_t^{\text{final}})$
      Mark F as requiring a repair step before termination
      if $\delta \geq D_{\max}$ **then**
        return $(I, \texttt{failed})$
      end if
      $\delta \leftarrow \delta + 1$
      continue
    end if
  end if
  Set success flag $z \leftarrow$ false, latest feedback $\phi^\star \leftarrow \emptyset$
  for $r = 1$ **to** K_{attempt} **do**
    if $b = 0$ or $z = \text{true}$ **then**
      break
    end if
    Append any retry hint in $\phi^\star$ to u, giving u_r
    Generate candidate $\hat{I} \leftarrow G_g(I, m, u_r)$
    Update remaining budget $b \leftarrow b - 1$
    $(p, \phi) \leftarrow \text{StepVerify}(\hat{I}, I, V, \Delta)$
    if $p = \text{true}$ **then**
      Commit candidate $I \leftarrow \hat{I}$
      Append a to accepted history A
      Reset degradation depth $\delta \leftarrow 0$
      Set $z \leftarrow \text{true}$
    else
      Reject candidate $\hat{I}$
      Set latest feedback $\phi^\star \leftarrow \phi$
    end if
  end for
  if $b = 0$ **then**
    break
  end if
  if $z = \text{false}$ **then**
    Set $F \leftarrow \text{StepFail}(a, \phi^\star)$
    if $\delta < D_{\max}$ **then**
      Mark the latest failure as requiring finer-grained decomposition of the current subgoal q
      Update degradation depth $\delta \leftarrow \delta + 1$
      continue
    else
      return $(I, \texttt{failed})$
    end if
  end if
end while
return $(I, \texttt{failed})$

Algorithm 1 allows up to K_{attempt} attempts per step, including the initial attempt. $G_g(I, m, u)$ invokes generator g with mode m, instruction u, and editing input I; $I = \emptyset$ denotes text-to-image generation. $\text{StepVerify}(\hat{I}, I, V, \Delta)$ checks candidate $\hat{I}$ against I, VPs V, and expected change Δ, returning pass flag p and feedback ϕ. $\text{FinalVerify}(I, V_t^{\text{final}})$ returns pass flag p and feedback ψ. Each

finish proposal supplies completion VPs; deterministic rules add missing perpendicularity/midpoint guards when applicable and a no-extra-content check. $\mathrm{StepFail}$ and $\mathrm{FinalFail}$ store failure records in F for subsequent planning. Final-failure records include the checked VPs, which may be reformulated in later finish proposals for the same task prompt.

When retries are exhausted or final verification rejects, the controller increments the degradation depth and requests replanning if $\delta < D_{\max}$. Replanning is allowed at depth $D_{\max}$; another exhausted retry loop or final rejection at that depth terminates execution. An accepted step resets the depth to zero. The implementation checks the generation budget before the next planner round: once it is exhausted, execution terminates without an additional final check, even if the last step was accepted. Human review evaluates the returned image; an empty image counts as task failure. Planner rounds are additionally capped at $3B$ to bound repeated finish calls.

A.7 Category-wise Verifier Performance

Table 7 reports category-wise balanced accuracy for VLM-based structured verification. The results show that verifier performance varies across categories, with geometry remaining challenging for most VLMs.

Table 7: Category-wise balanced accuracy of VLM-based verifiers.

VLM	Geometry	Circuit	Flowchart	General
Qwen3-VL-8B-Instruct	54.18	55.34	56.35	55.66
Qwen3-VL-32B	55.11	54.64	56.94	61.48
InternVL3-8B	51.82	60.75	63.69	61.17
Gemini-3.1-Flash-Lite	61.41	67.88	70.22	71.52
Gemini-3.1-Pro-Preview	82.50	70.74	70.59	74.02

A.8 Baseline Implementations

This section describes the baselines used in the hardest-subset experiments. Since prior agentic image-generation systems target different input-output settings, we implement adapted baselines that preserve their core planning or reflection principles rather than direct reproductions. All methods use the same target image generator, canvas setting, and human-review protocol for TSR; runtime verification follows each method's controller. Fixed-K baselines use exactly K generation/editing attempts, while X-Planner-inspired, GRA, and ProGraf are constrained by the same global attempt budget B. The baselines test whether static decomposition, structured planning, or generic reflection is sufficient without generator-specific profiling and VP-grounded control. Table 8 summarizes the main differences.

Fixed-K static baselines. The Fixed-K baselines test whether simple static decomposition is sufficient without closed-loop control. Given a task prompt, the planner decomposes it once into exactly K steps, where we evaluate $K = 3$ and $K = 5$. The first step is executed as text-to-image generation, and subsequent steps are executed as image edits. Since every run executes exactly K generation/editing calls, its total attempts are fixed by design, i.e., $\mathrm{ATT} = K$. The method does not perform step-level verification, retry, accepted-state commitment, or online replanning. A final verifier checks the output, and human review determines reported TSR.

X-Planner-inspired static baseline. This baseline adapts the structured-action planning idea of X-Planner [36] to our structured diagram setting. It generates a complete multi-step action plan before execution. Each step contains a structured local action, including an operation type, local goal, target anchors, preserve anchors, editing instruction, expected change, and step-level VPs. During execution, a step-level VLM gate checks whether the current step succeeds and returns a binary success/failure signal. If the step fails, the same step can be retried until it passes or the global budget is exhausted. However, the plan itself is static: the method does not use generator profiles, online replanning, or degradation.

Table 8: Summary of baseline and planner mechanisms. All methods use the same target generator, canvas setting, and human-review protocol for TSR. Fixed-K baselines use exactly K steps, while X-Planner-inspired, GRA, and ProGraf share the same global attempt budget B.

Mechanism	Fixed-K	X-Planner	GRA	ProGraf
Static full-plan decomposition	Yes	Yes	No	No
Dynamic next-step planning	No	No	Yes	Yes
Generator-specific profile	No	No	No	Yes
Step-level feedback	No	Binary gate	Free-form critic	VP verifier
Explicit failed-VP feedback	No	No	No	Yes
Step retry	No	Yes	Yes	Yes
Online replanning	No	No	Yes	Yes
Degradation	No	No	Generic	Profile-guided
Continue after final failure	No	No	Yes	Yes

Generic reflective agent baseline. GRA is an adapted generic reflective agent inspired by the feedback-driven refinement principle of Talk2Image [18]. At each round, it parses the current state, plans the next action, executes it, and receives feedback from a free-form VLM critic. The critic feedback can trigger retry or replanning, and final-verification failure is written back into the interaction history if budget remains. Unlike ProGraf, GRA does not use explicit VP-based step verification, generator-specific profiles, or profile-guided degradation. Thus, it tests whether a generic reflective closed loop is sufficient without structured VP-grounded control.

ProGraf. ProGraf uses the same target generator and global budget as the agent baselines, but differs in two key aspects. First, it conditions planning on generator-specific profiles extracted from benchmark records, including category-level reliability, tag-level tendencies, and recurrent failure patterns. Second, it uses VP-grounded step verification as a structured control signal: only verifier-approved intermediate states are committed, while failed candidates are rejected and converted into failure context for retry, degradation, or replanning. In implementation, the planner prompt also includes fixed format demonstrations to standardize the action schema and VP format. These demonstrations are task-independent and do not contain generator outcomes, reviewed failure notes, or prompt-specific solutions. Therefore, ProGraf tests whether generator profiling and VP-grounded closed-loop control provide additional gains beyond static decomposition and generic reflection.

A.9 Compute Resources

Most closed-source generator and VLM experiments are API-based, so the provider-side hardware configuration is not visible to us. For open-source models, we run local experiments on a server with an Intel(R) Xeon(R) Gold 6426Y CPU and $4\times$ NVIDIA A100-SXM4-80GB GPUs. This local setup is used for open-source VLM verifier evaluation, including Qwen3-VL-8B-Instruct, Qwen3-VL-32B, and InternVL3-8B, and for open-source image-generation backends such as GLM-Image and FLUX2-9B. The open-source VLMs are served locally for VP-level verification, while the open-source image generators are exposed as local image generation/editing HTTP endpoints and called by the same evaluation, ProGraf, and baseline code paths when applicable. All experiments use the same square canvas setting and the same evaluation protocol. Planner and step-verifier calls use a 240s timeout, and final verification uses a 420s timeout.

A.10 Broader Impact

This work aims to improve the reliability and verifiability of structured visual generation, which may benefit education, technical documentation, and diagram authoring. At the same time, unreliable structured generation could mislead users in settings where diagrams carry procedural, engineering, or instructional meaning. We therefore emphasize verification, human-reviewed evaluation, and explicit reporting of failure modes. We do not advocate deploying generated diagrams in safety-critical settings without domain-specific validation.

B Additional Analysis on Profiling and Budget

B.1 Profiling Ablation

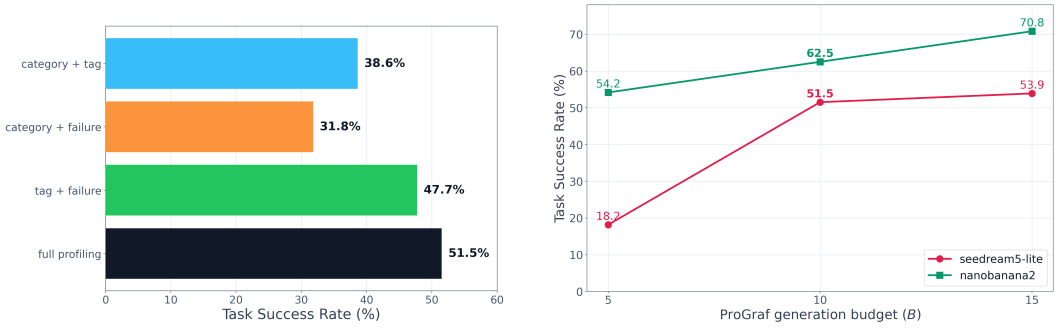
Table 9 compares ProGraf with and without generator-specific profiling on the geometry hardest subset. Removing profiling reduces task success rate from 51.52% to 30.30% for seedream5-lite and from 62.50% to 54.17% for Nanobanana2. This indicates that the gains are not explained by retry and replanning alone; benchmark-derived profiles provide useful priors for decomposition granularity and operation ordering.

Table 9: Profiling ablation on the geometry hardest subset. Removing generator-specific profiling reduces task success rate for both tested generators.

Model	Setting	Avg. TSR
seedream5-lite	w/ profiling	51.52%
seedream5-lite	w/o profiling	30.30%
Nanobanana2	w/ profiling	62.50%
Nanobanana2	w/o profiling	54.17%

B.2 Profiling Components and Budget Sensitivity

Figure 5 provides two complementary analyses on the geometry hardest subset. Figure 5(a) studies profiling component robustness on seedream5-lite. The full profile achieves the best performance, while tag-level tendencies combined with recurrent failure patterns provide the strongest partial profile. Figure 5(b) reports budget sensitivity.



(a) Profiling component robustness.

(b) Budget sensitivity.

Figure 5: Profiling component and budget analyses on geometry hardest subsets. (a) Profiling component robustness on seedream5-lite. Category, tag, and failure correspond to C_g, T_g, and F_g, respectively. Each partial profile keeps two components and removes the remaining one. (b) Budget sensitivity under different generation budgets.

B.3 Accuracy–Efficiency Trade-off

Figure 6 compares task success rate and average attempts used on the geometry hardest subset. ProGraf generally achieves higher task success than fixed-step and agent baselines, while keeping the number of generation or editing attempts bounded. This supports the conclusion that the improvement is not simply due to unconstrained retry, but to more effective use of the available budget.

B.4 Qualitative Failure Analysis

Figure 7 shows two representative failure cases of ProGraf on geometry tasks. Although ProGraf decomposes the target diagram into verified intermediate steps, some geometric relations remain difficult when the underlying generator cannot reliably execute the required local operation.

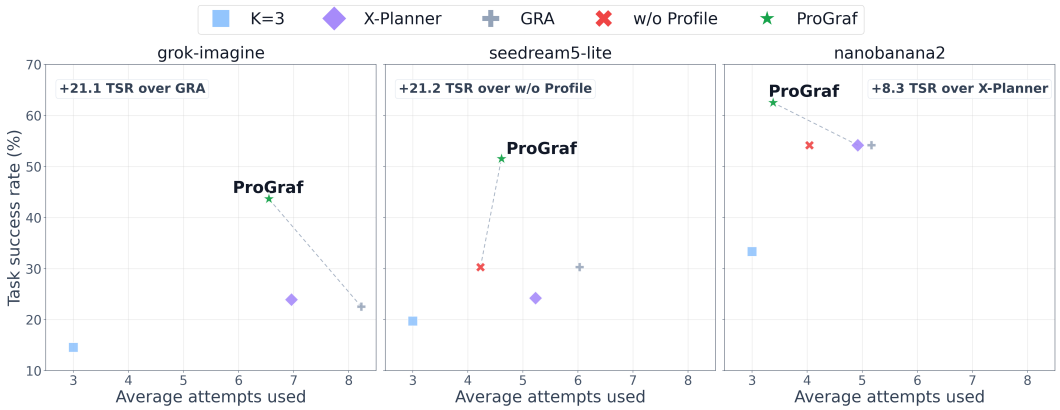


Figure 6: Accuracy–efficiency trade-off on the geometry hardest subset. Higher task success rate and fewer attempts are preferred.

Unstable perpendicular relations. In Case 1, the task requires two perpendicular relations: $AD \perp BC$ and $BE \perp AC$. ProGraf already isolates the construction of perpendicular segments into separate steps, rather than asking the generator to satisfy all constraints in one shot. However, the final candidate still fails because segment BE is not visually perpendicular to AC. Notably, the generated image includes a perpendicular marker, but the actual drawn geometry is visibly skewed. This illustrates a common failure mode: generators may produce relation symbols without accurately satisfying the corresponding geometric relation.

Unstable midpoint construction. Case 2 shows a failure involving a midpoint constraint. The task requires point M to be the midpoint of segment BC, followed by drawing segment ME. After repeated failures, ProGraf triggers degradation and decomposes the operation into finer-grained subgoals, first marking M on BC and then connecting E and M. Nevertheless, the generated point M is still not reliably placed at the midpoint of BC. This suggests that midpoint construction is another unstable primitive for current generators: even when the operation is isolated and ordered carefully, the model may still fail to execute it precisely.

Limitations of closed-loop orchestration. These cases highlight a boundary of our method. ProGraf improves generation by controlling step granularity, execution order, retry, and degradation, but it does not directly improve the primitive drawing capability of the generator. When a generator is intrinsically weak at operations such as perpendicular construction or midpoint placement, closed-loop planning can reduce the difficulty of each step but cannot always guarantee success.

Verifier sensitivity. These failures also expose a limitation of VLM-based verification. For perpendicular relations, existing VLMs often rely too heavily on explicit relation symbols, e.g., right-angle marks, and may accept diagrams where the marker is present but the actual geometry is not perpendicular. Although our verifier prompt explicitly instructs the VLM to judge geometric relations from the drawn geometry rather than from symbols alone, this only partially mitigates the problem. Midpoint relations pose a similar challenge: the verifier may struggle to determine whether a marked point is truly centered on a segment. The strictness of relation verification therefore strongly affects the whole closed-loop process. If the verifier is too strict, many usable intermediate states are rejected and the process may fail to make progress; if it is too loose, erroneous candidates may be accepted and propagated to later steps.

B.5 Leave-geometry-out profile transfer

To test whether the profile merely memorizes geometry-specific benchmark failures, we extract profiles using only circuit, flowchart, and general-diagram records, and evaluate ProGraf on the geometry hardest subset. As shown in Table 10, removing geometry records from profile construction reduces performance compared with the full-profile setting, which is expected because geometry-specific relation failures are no longer available. However, the leave-geometry-out profile does

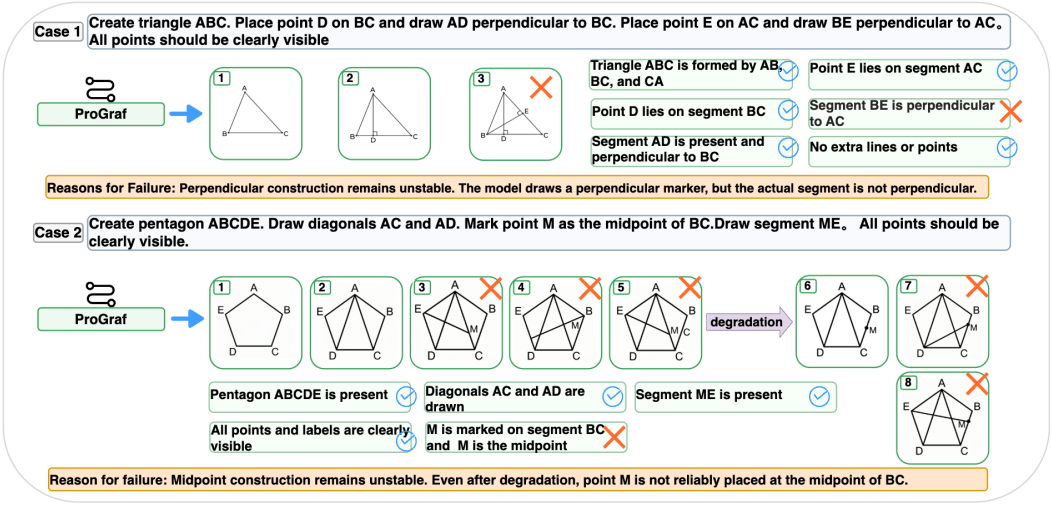


Figure 7: Representative failure cases of ProGraf. Case 1 shows unstable perpendicular construction: although ProGraf isolates perpendicular drawing into separate steps, the generated segment BE is still not perpendicular to AC, despite the presence of a perpendicular marker. Case 2 shows unstable midpoint construction: even after degradation into finer-grained subgoals, the generator fails to place M reliably at the midpoint of BC. These cases show that closed-loop orchestration can adjust step order and granularity, but remains bounded by the generator's primitive geometric capabilities and by the reliability of VLM-based relation verification.

Table 10: Leave-geometry-out profile transfer on the geometry hardest subset. The leave-geometry-out profile is extracted only from circuit, flowchart, and general-diagram records, excluding all geometry records. TSR is reported as percentage.

Model	Profile setting	Profile source	TSR	Att.
grok-imagine	Full profile	Geo+Cir+Flo+Gen	43.7	6.55
	Leave-Geom-Out	Cir+Flo+Gen	36.6	6.82
seedream5-lite	Full profile	Geo+Cir+Flo+Gen	51.5	4.61
	Leave-Geom-Out	Cir+Flo+Gen	36.4	5.82

not collapse: it still achieves 36.62% TSR on grok-imagine and 36.36% TSR on seedream5-lite. These results suggest that the profile captures transferable generator tendencies, such as editing instability, extra-element errors, and relation-construction difficulty, rather than simply memorizing target-category records.

C Held-Out Profile Evaluation and Controls

C.1 Prompt-Cluster Split and Profile Construction

The supplementary held-out experiment uses seedream5-lite and all 154 Geometry prompts. We normalize predefined boilerplate, case, punctuation, and symbolic point/line labels. Prompt pairs with normalized token-Jaccard similarity of at least 0.75 are connected, and each connected component is assigned wholly to one fold. This deterministic construction produces 110 prompt clusters and fold sizes of 31/31/31/31/30. Fold construction uses prompt text only; VPs, images, generator outcomes, failure notes, and hardest-set membership do not determine the split. The clusters identify high-overlap surface variants under this normalization.

For evaluation fold f, all three historical seedream5-lite output records for every prompt in f are removed before applying the profile extractor Φ. Held-out images, VP labels, task outcomes, false VPs, and reviewed notes therefore contribute no information to that fold's profile. Calibration uses the other four Geometry folds and all non-Geometry records. Under the same normalization, the

maximum similarity between a retained non-Geometry prompt and a Geometry prompt is 0.214. Applying the original all-three-fail criterion inside each held-out fold yields 7/4/2/3/6 hard prompts, covering the same 22 unique hardest prompts once per run. Fold results are pooled after their profiles have been fixed.

C.2 Repeated Runs and Prompt-Level Uncertainty

Table 11: Three held-out Full ProGraf runs on seedream5-lite Geometry. The mean TSR uses the same 22 prompts in each run; $\pm$ denotes sample SD across the three run-level TSRs.

Run	Human-correct outputs	TSR (%)	ATT
r1	11/22	50.00	5.05
r2	10/22	45.45	6.18
r3	11/22	50.00	5.50
Mean	32/66 task-runs	48.48 ± 2.62	5.58

The held-out mean is 3.04 percentage points below the original full-profile result of 51.52%. Repetitions characterize generation stochasticity over 22 unique prompts. For prompt-bootstrap uncertainty, we average the three binary outcomes within each prompt and resample the 22 prompts with replacement for 10,000 bootstrap repetitions. The 2.5th and 97.5th percentiles give a 95% interval of [31.82%, 65.15%]. This interval and the sample SD describe different sources of variation: uncertainty over prompts and variation across runs, respectively.

C.3 Budget-Matched and Component Control Definitions

All conditions in Table 3 use the same 22 prompts, seedream5-lite backend, Gemini-3.1-Pro-Preview verifier, and maximum generation budget $B = 10$. Conditions using a planner use the same Gemini model and a fixed, task-independent Geometry VP-format template. Conditions retaining a profile use the fold-specific held-out profile. Full uses a per-step attempt limit $K_{\text{attempt}} = 3$ and maximum degradation depth $D_{\max} = 2$. Final TSR is determined by human review. The controls below each have one run; Full has three.

Best-of-10. Ten independent text-to-image candidates are generated from the original prompt, without image editing or profile guidance. The same Gemini verifier checks each against the benchmark VPs. Selection prioritizes a final-verifier pass, then the number of satisfied VPs, then earliest generation order for ties. Human review of the selected image yields 7/22 successes (31.82%). The prompt-bootstrap 95% interval is [13.64%, 50.00%]. For a paired comparison, we subtract each prompt's mean outcome over the three Full runs from its single Best-of-10 outcome and bootstrap prompts; the mean difference is -16.67 percentage points with interval $[-36.36, 3.03]$. Selection uses the verifier rather than an oracle human choice among the ten candidates.

w/o VP grounding. The verifier retains a binary accept/reject gate but receives no explicit VP list. Retry and replanning receive no VP labels, failure reason, or retry hint. Human review yields 7/22 successes (31.82%).

w/o rollback. After rejection, the failed candidate becomes the image used for retry and replanning instead of the last committed image. Human review yields 4/22 successes (18.18%).

Retry-only. Profile guidance and VP verification remain enabled. A failed instruction is repeated until it succeeds or $B = 10$ is exhausted, with verifier retry hints permitted. Failure-driven replanning and degradation are disabled. Human review yields 6/22 successes (27.27%).

Profile-only. The planner retains profile guidance and observes the current image, but candidates are directly committed. Runtime verification, retry, rollback, and failure-driven replanning are disabled. Human review yields 6/22 successes (27.27%).

The separate three-run profiling ablation in Appendix B removes P_g while retaining the closed loop and obtains 30.30% TSR on seedream5-lite Geometry. Its full-profile reference is 51.52%.

C.4 Concrete Profile and Planner Trace

Task T109 requires triangle ABC, $AD \perp BC$, and $BE \perp AC$. Its actual held-out profile contains the entries in Table 12. The strong tags are global, nonexclusive prompt-derived tags in the serialized profile; the Geometry statistic, weak tags, and perpendicular/line failure patterns are the task-relevant entries for this case.

Table 12: Selected entries from the seedream5-lite held-out profile used for T109. Support n counts unique calibration prompts. Success rates use their $3n$ reviewed outputs; failure counts are matched false-VP occurrences.

Field	Entry	Success (%)	Support n / count
Category	Geometry	63.14	123
Global strong tag	geometry	78.57	42
Global strong tag	sequence	75.90	65
Global strong tag	flowchart	74.85	57
Weak tag	triangle	43.86	57
Weak tag	line_relation	57.97	46
Geometry failure	perpendicular	—	15
Geometry failure	segment_or_line	—	48

For example, the Geometry rate is $233/369 = 63.14\%$, using three reviewed outputs for each of 123 calibration prompts. These entries are extracted from calibration records excluding T109's fold. The planner's recorded reason cites difficulty with complex triangle compositions and dependent relations. It first constructs a spacious acute triangle, then adds $AD \perp BC$, and then separately adds $BE \perp AC$ while preserving ABC, AD, and D. The first BE candidate is rejected and its retry accepted. The final four-call image is human-correct. This trace illustrates a construction order consistent with the profile's relation weaknesses.

C.5 Verifier Errors and Their Trajectory Impact

The confusion matrix underlying the submitted 76.89% balanced accuracy has 14,029 true positives, 415 false positives, 973 true negatives, and 2,735 false negatives at the VP level. Its false-accept rate among human-negative VPs is $415/(415 + 973) = 29.90\%$, and its false-reject rate among human-positive VPs is $2735/(14029 + 2735) = 16.31\%$. These rates describe the offline VP benchmark. To examine effects during execution, we audit all 34 human-incorrect trajectories from the three held-out Full runs.

Among 66 task-runs, 51 finish with a final-verifier pass: 32 are human-correct and 19 are human-incorrect. The other 15 terminate unsuccessfully and are all human-incorrect. One records a final-verifier rejection and 14 never reach final verification. Thus, budget exhaustion is distinguished from an explicit final-verifier rejection.

All 19 failures ending in a system-accepted final image contain an incorrect intermediate state accepted by the verifier and propagated into the final failure. Of the 15 budget-exhausted failures, two (r1/T140 and r1/T148) contain a usable intermediate state that was falsely rejected; subsequent trajectories exhaust the budget and remain incorrect. Manual review identifies no contributing verifier error in the other 13 budget-exhausted failures. Identified verifier errors therefore contribute to 21 of the 34 failed trajectories. The audit describes observed effects on failed trajectories; its denominator is failed task-runs rather than all executed candidates.

T108 illustrates error propagation on midpoint relations. Across the three original seedream5-lite outputs, midpoint P is incorrect three times and N twice. In held-out r1, the verifier accepts the local midpoint checks and the final image, while human review rejects the actual midpoint relation. The perpendicular case in Figure 7 is analogous: a right-angle marker is present while the segments remain skewed. These relations challenge both generation and verification. Accepted-state rollback restores the image after a rejected edit, but an incorrect state that passes verification can remain committed and influence later steps.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the benchmark, verifier evaluation, and ProGraf claims, and the experimental sections report the corresponding generator, verifier, hardest-subset, and ablation results.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The conclusion discusses the scope (recovery for black-box raster generators, not symbolic/vector synthesis), the need for reviewed calibration records, the limited held-out evidence (one generator, one category, 22 hard prompts, single-run controls), conditional hardest-subset evaluation, dependence on the generator's primitive capabilities, and verifier error propagation; Appendix C.5 and the qualitative failure analysis give further detail.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present formal theoretical results or proofs; it provides an algorithmic framework and empirical evaluation.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The experimental setup, benchmark construction, model choices, hardest-subset protocol, baselines, budgets, verification procedure, and metrics are described in the main text and appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Experimental details are provided in Appendices A–C. We plan to release the code and benchmark upon publication to support reproducibility.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies the evaluated models, data construction protocol, repeated-run setting, hardest-subset definition, planner budget, retry/degradation parameters, baseline settings, and evaluation metrics.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Figure 3 reports error bars for the main hardest-subset results after adding standard deviations over repeated runs when available. The paper also reports raw counts for small subsets.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We report compute resources and runtime settings in Appendix A.9. Closed-source generators and Gemini-based VLMs are accessed through APIs, while open-source VLMs and image-generation backends are evaluated on a server with an Intel(R) Xeon(R) Gold 6426Y CPU and 4× NVIDIA A100-SXM4-80GB GPUs.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research follows the NeurIPS Code of Ethics: it studies synthetic structured-diagram prompts and generated images, does not use private personal data, and reports evaluation and limitations transparently.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses broader impacts in Appendix A.10, including potential benefits for structured visual authoring and risks from unreliable generated diagrams in procedural or technical settings.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a high-risk pretrained model, scraped image dataset, or unrestricted image generator. The benchmark consists of structured text prompts and verification points.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Existing models and tools used in the experiments are named and cited where applicable. Access to closed-source generators and VLMs follows their respective API terms.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The paper introduces a new benchmark of structured non-natural image tasks and documents its categories, task construction, VP types, annotation/review protocol, and evaluation metrics.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing experiments or human-subject studies. Human review is used only to validate generated images and VP correctness for the benchmark evaluation.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human-subject research or crowdsourcing studies requiring IRB approval.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs and VLMs are core components of the research: they are used for planning, step-level and final verification, and parts of benchmark-assisted evaluation. Their roles are described in the method and experimental setup.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.