LEARNING FROM MODEL FAILURES DISTILLED AS EXECUTABLE VISUAL PROGRAMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Text-to-image (T2I) models can often satisfy a generation requirement on some inputs while failing on closely related ones, suggesting that useful capabilities may already be present but are expressed inconsistently. Preference-based alignment provides a natural mechanism for reinforcing desired generations and discouraging undesirable ones. However, its effectiveness depends not only on how preferences are optimized, but also on the informativeness of the preferred–rejected contrast. We introduce **Failure-Guided Preference Supervision (F-GPS)**, a framework that constructs preference supervision from failures observed in the target T2I model. Rather than treating failures only as diagnostic signals, F-GPS *distills recurring failure patterns into reusable programs*: it atomizes observed coarse failure instances into fine-grained visual factors, aggregates these factors across samples into a candidate inventory, filters the inventory into reusable atomic visual cues, and operationalizes the retained cues as parameterized failure programs. A randomly sampled composition of these failure programs is then applied to an editable preferred example to construct a rejected example with targeted deviations, yielding an informative preference contrast. We study F-GPS on *image composition* and *visual text rendering*, two settings exhibiting distinct forms and granularities of generation failure. Across multiple T2I backbones and preference-optimization objectives, F-GPS improves downstream alignment, with gains of up to 5.6 pp for image composition and 10.4 pp for visual text rendering over SFT. These results suggest that failure patterns observed in the target model can provide targeted supervision for preference-based T2I alignment.

1 INTRODUCTION

Pretrained T2I models (Podell et al., 2023; Rombach et al., 2022; Esser et al., 2024) have improved substantially across successive generations. Nevertheless, their behavior remains inconsistent: the same model may satisfy a generation requirement in one instance yet fail on a closely related one (Hayes et al., 2025; Vemishetty et al., 2025). Such instance-level inconsistency suggests that improving reliability may not always require learning an entirely new behavior from scratch. This motivates post-training approaches that reinforce desired generations, which have been shown to improve alignment and generation quality in pretrained diffusion models. Preference optimization provides a natural mechanism for this purpose by reinforcing preferred generations over undesirable alternatives (Wallace et al., 2024; Zhu et al., 2025). Its effectiveness, however, depends not only on how the preference signal is optimized, but also on the contrast from which that signal is derived (Croitoru et al., 2025; Hu et al., 2025).



Figure 1: Representative F-GPS preference pairs for **image composition** (top) and **visual text rendering** (bottom).

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

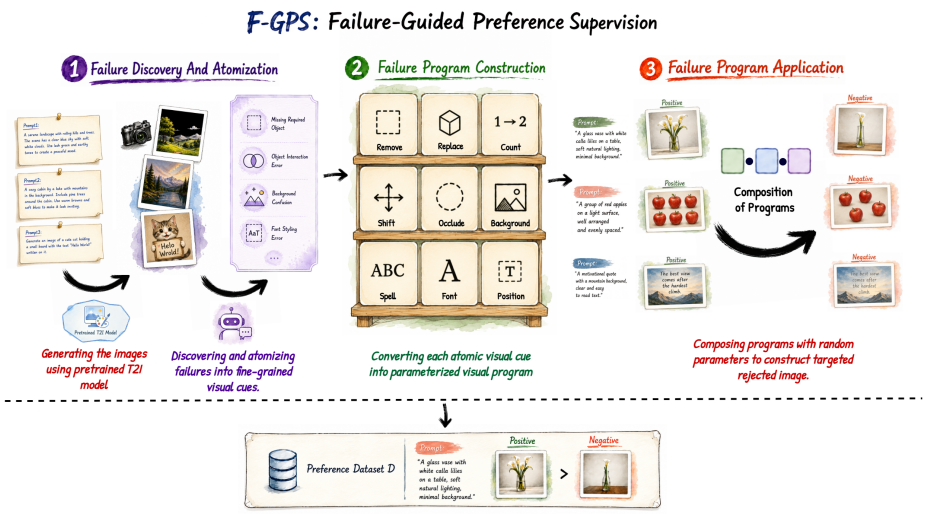


Figure 2: **Architecture of the F-GPS framework.** F-GPS distills structured failures from the target T2I model into failure programs and applies randomly sampled program compositions to editable preferred examples to construct targeted rejected images for preference optimization.

Most preference learning methods (Wallace et al., 2024; Zhu et al., 2025; Zhang et al., 2024) primarily focus on the optimization formulation, while rejected examples are often obtained from existing preference data or model-generated outputs. When model-generated samples are used directly as rejected examples (Zhang et al., 2024; Hu et al., 2025), the specific undesirable behavior that should distinguish the rejected image from the preferred one is not explicitly controlled, and multiple visual differences may be entangled within the same comparison. Recent work has therefore begun to treat preference-pair construction more explicitly, using visually consistent fusion, real-image anchors, or object-centric prompt perturbations (Hu et al., 2025; Chen et al., 2026; Oh et al., 2025). These approaches reinforce that the source and structure of the preference pair can materially affect the supervision provided to the model.

This raises a central question: *what should guide the construction of the rejected example in an informative preference pair?* We consider failures exhibited by the target model itself as a natural source of this information. Among the broad range of T2I failures, we focus on *structured failures*: deviations from the prompt that can be localized to explicit visual requirements, rather than being characterized only by overall perceptual quality. Prior work documents such systematic failure dimensions across T2I models and evaluation settings. TIFA (Hu et al., 2023) and T2I-CompBench (Huang et al., 2023) reveal persistent limitations in counting, attribute binding, spatial relations, and multi-object composition, while TextDiffuser (Chen et al., 2023) highlights systematic difficulties in rendering textual content accurately. These limitations also persist in newer variants, which continue to exhibit various fine-grained failures (Hayes et al., 2025; Vemishetty et al., 2025). Taken together, these observations suggest that many generation failures have recurring patterns, making the target model's observed failures a useful basis for determining which targeted deviations to introduce when constructing rejected examples.

Building on this observation, we introduce **F-GPS**, a framework that distills recurring patterns from structured failures observed in the target T2I model into reusable programs for constructing preference contrasts. Given outputs from the target model, a failure analyst identifies sample-specific *coarse failure instances*, atomizes them into *fine-grained visual factors*, and aggregates the resulting factors across samples to form a candidate inventory. The analyst then filters this inventory by removing redundant, vague, or non-programmable factors, yielding reusable *atomic visual cues*. Each retained cue is operationalized as a parameterized *failure program*. Randomly sampled compositions of these failure programs are then applied to editable preferred examples to construct rejected examples with targeted deviations for preference optimization (Figures 1 and 2).

Structured T2I failures can manifest in qualitatively different forms, such as object-level compositional errors and fine-grained text-rendering errors. We therefore evaluate F-GPS on *image com-*

position and *visual text rendering* as two complementary settings for testing whether the same failure-guided supervision framework remains effective across distinct types and granularities of failure. Across our experiments, we evaluate F-GPS on multiple T2I backbones, including SD v1.5, SDXL-Base-1.0, and SD3.5-Medium, and under multiple preference objectives, including Diffusion-DPO (Wallace et al., 2024), Linear-DPO (Li et al., 2026), and DSPO (Zhu et al., 2025).

Our key contributions are as follows:

- **Failure-guided preference supervision.** We introduce F-GPS, a framework that uses structured failures observed in a target T2I model to guide the construction of preference supervision.
- **Failure distillation into reusable programs.** We distill recurring failure patterns into parameterized failure programs and apply randomly sampled program compositions to editable preferred examples to construct rejected examples with targeted deviations for preference optimization.
- **Evidence across distinct structured-failure settings.** We evaluate F-GPS on image composition and visual text rendering across multiple T2I backbones and preference objectives, showing consistent improvements across these settings.

2 Related Work

2.1 Failure Analysis in Text-to-Image Models

Despite rapid improvements in overall image quality, T2I models continue to exhibit diverse and structured generation failures. FineGRAIN (Hayes et al., 2025) organizes T2I behavior into a fine-grained taxonomy of failure modes spanning object attributes, counting, spatial relations, anatomy, physical relations, and text rendering. Compositional benchmarks such as T2I-CompBench (Huang et al., 2023), TIFA (Hu et al., 2023), and GenEval (Ghosh et al., 2023) expose persistent errors in attribute binding, object counting, spatial and non-spatial relations, object co-occurrence, and multi-object composition. Visual text generation presents a different set of challenges: TextDiffuser (Chen et al., 2023) highlights difficulties in rendering accurate and coherent text, while LenCom-Eval (Lakhanpal et al., 2025) further examines failures involving lengthy and complex text sequences. Recent evaluations further show that such prompt-adherence failures persist in newer T2I models, including for simple geometric and spatial requirements (Vemishetty et al., 2025). Complementary work moves beyond predefined benchmarks to actively discover failures: SAGE (Liu et al., 2024b) searches prompt and latent spaces for systematic undesirable behaviors, while LLM-assisted red teaming (Sagar et al., 2024) uses reinforcement-learning-based exploration together with LLM-generated rewards and states to characterize failure landscapes in diffusion models. Collectively, these studies show that T2I failures span diverse yet identifiable forms that can be systematically characterized. F-GPS builds on this observation by using failures observed in the target model not only for evaluation or diagnosis, but to guide the construction of preference supervision.

2.2 Programmatic Supervision and Controlled Data Construction

Programmatic supervision provides a way to encode reusable knowledge in executable programs rather than relying solely on instance-wise annotation. Data programming (Ratner et al., 2016) formalizes domain heuristics as labeling functions that automatically generate weak supervision, while Kaushik et al. (2019) and Gardner et al. (2020) use controlled edits to isolate task-relevant distinctions in training and evaluation, respectively. In vision, systems such as VisProg (Gupta & Kembhavi, 2023) and ViperGPT (Surís et al., 2023) use LLM-generated programs to compose visual tools, while Visual Program Distillation (Hu et al., 2024) uses verified executable programs as an intermediate mechanism for synthesizing reasoning supervision for vision-language models (VLMs). Programmatic rendering has likewise been used to construct visual training data, from synthetic scene-text generation (Gupta et al., 2016) to recent large-scale text-rich data construction such as TextSculptor (Lin et al., 2026). F-GPS builds on this idea by *distilling recurring failure patterns observed in the target model into executable failure programs*. These programs operationalize the distilled failure knowledge as transformations that introduce targeted deviations into editable preferred examples, yielding rejected examples for preference alignment. The contribution therefore lies not in rendering or compositing itself, but in distilling observed model failures into reusable programs for constructing preference supervision. See Appendix B for extended related work.

3 METHODOLOGY

We present **F-GPS**, a failure-guided preference supervision framework for aligning T2I models. F-GPS distills recurring failure patterns observed in the target model into reusable parameterized failure programs, which are subsequently applied to editable preferred examples to construct targeted preference supervision (Figure 1). The framework consists of three stages (Figure 2): **(1) Failure Discovery and Atomization**; **(2) Failure Program Construction**; and **(3) Failure Program Application**.

Let $\mathcal{C}$ denote the prompt space, $\mathcal{X}$ the image space, and $\mathcal{Z}$ an editable structured representation space whose elements explicitly expose visual elements and their attributes so that they can be independently modified before rendering. For example, an image may be represented by a collection of editable elements together with attributes such as position, scale, orientation, color, appearance, or textual content. Let $R : \mathcal{Z} \to \mathcal{X}$ denote the corresponding renderer, and let $G_\theta : \mathcal{C} \to \mathcal{X}$ denote the target T2I model whose outputs are analyzed for structured failures. We denote the set of editable preferred examples by $\mathcal{D}^+ = \{(c, z^+)\}$, where $c \in \mathcal{C}$ is the associated prompt and $z^+ \in \mathcal{Z}$ denotes the editable structured representation of the preferred example. Rendering z^+ through R produces the corresponding preferred image $x^+ = R(z^+)$. preferred image $x^+ = R(z^+)$. We further denote by $\mathcal{E}$ a failure analyst capable of interpreting visual content, reasoning over visual evidence, and understanding relationships among image elements. In our experiments, $\mathcal{E}$ is instantiated with GPT-4o. The analyst examines outputs of G_θ and assists in discovering and atomizing failures, filtering the resulting visual factors, and constructing their executable representations. All instructions given to the failure analyst, together with the seed prompts used for failure discovery, are provided in the Appendix J and L.

3.1 FAILURE DISCOVERY AND ATOMIZATION

We begin with a seed prompt set $\mathcal{C}_{\text{seed}} = \{c_1, \ldots, c_N\} \subseteq \mathcal{C}$ and generate corresponding outputs $\tilde{x}_i = G_\theta(c_i)$. For each generated image $\tilde{x}_i$, the failure analyst $\mathcal{E}$ examines the image together with its prompt c_i and identifies visible deviations from the intended generation. We refer to these sample-specific observations as *coarse failure instances*. A coarse failure instance captures the error as observed in a particular generated sample and may subsume multiple visual deviations. For example, *text misalignment* may jointly reflect horizontal or vertical displacement, rotation, or irregular spacing.

For a generated image $\tilde{x}_i$, let

$$\mathcal{F}_i = \{f_{i,1}, \ldots, f_{i,M_i}\}$$

denote the set of coarse failure instances identified in that sample.The failure analyst $\mathcal{E}$ then *atomizes* each instance into fine-grained visual factors,

$$\mathcal{A}(f_{i,j}) = \{a_{i,j,1}, \ldots, a_{i,j,L_{i,j}}\},$$

where each $a_{i,j,k}$ denotes a visual factor that can be independently described and manipulated. Atomization therefore moves from a sample-specific description of *what went wrong* to the constituent visual factors through which the failure is manifested. For example, a coarse *text-misalignment* failure may be atomized into horizontal offset, vertical offset, angular rotation, and irregular spacing.

Aggregating the resulting factors across analyzed samples gives the candidate inventory

$$\mathcal{A}_{\text{cand}} = \bigcup_{i=1}^{N} \bigcup_{f \in \mathcal{F}_i} \mathcal{A}(f).$$

Because similar factors may arise across different samples or under different descriptions, the failure analyst $\mathcal{E}$ filters this inventory by removing redundant, vague, or non-programmable factors. This yields the retained set of reusable *atomic visual cues*

$$\mathcal{A} = \{a_1, \ldots, a_L\}, \qquad a_\ell \in \mathcal{A}_{\text{cand}} \;\; \forall \ell \in \{1, \ldots, L\}.$$

Representative coarse failure instances and their corresponding fine-grained visual factors are provided in the Appendix D.

3.2 Failure Program Construction

Following failure atomization and filtering, the failure analyst $\mathcal{E}$ converts each retained atomic visual cue $a_\ell \in \mathcal{A}$ into an executable *failure program*. Each program is represented by a parameterized transformation

$$T_\ell : \mathcal{Z} \times \Lambda_\ell \to \mathcal{Z},$$

where Λ_ℓ specifies the program's controllable parameters and their admissible ranges, which govern the severity and variation of the transformation.

```python
def horizontal_misalignment(example,
    parameters=None):
    """Deviate characters along the x-axis."""

    if parameters is None:
        parameters = {"max_x_offset": 10}

    new = copy.deepcopy(example)
    max_x_offset = parameters["max_x_offset"]

    for i, text in enumerate(new["text"]):
        if text:
            text_length = len(text)
            x_offsets = [random.uniform(
                -max_x_offset, max_x_offset)
                for _ in range(text_length)]
            new["x_offset"][i] = x_offsets

    return new
```

Figure 3: Horizontal misalignment transformation.

To construct a program, the failure analyst $\mathcal{E}$ is provided with the definition of the atomic cue together with metadata describing the editable representation, available attributes, and rendering interface. Based on this information, the analyst formulates an executable transformation that operationalizes the corresponding atomic visual cue within $\mathcal{Z}$ and defines suitable controllable parameters and their admissible ranges. The failure analyst $\mathcal{E}$ subsequently critiques and refines the candidate programs to improve their consistency with the intended atomic cue and compatibility with the underlying rendering interface. The resulting collection of parameterized programs forms the failure-program library

$$\mathcal{T} = \{(T_\ell, \Lambda_\ell)\}_{\ell=1}^{L}.$$

The resulting library provides a reusable executable representation of the recurring failure patterns distilled from the analyzed model outputs. Additional examples of the induced failure programs are provided in Appendix E.

3.3 Failure Program Application

Given an editable preferred example $(c, z^+) \in \mathcal{D}^+$, we first render the corresponding preferred image $x^+ = R(z^+)$. We then randomly sample r failure programs from $\mathcal{T}$. For each selected program T_{ℓ_k}, we sample a concrete parameter configuration $\lambda_k \in \Lambda_{\ell_k}$, yielding the instantiated transformation

$$T_{\ell_k,\lambda_k}(z) := T_{\ell_k}(z, \lambda_k), \qquad T_{\ell_k,\lambda_k} : \mathcal{Z} \to \mathcal{Z}.$$

Let $\pi = ((\ell_1, \lambda_1), \ldots, (\ell_r, \lambda_r))$ denote the resulting program composition. The corresponding composed transformation is

$$T_\pi = T_{\ell_r,\lambda_r} \circ \cdots \circ T_{\ell_1,\lambda_1}.$$

Program composition allows multiple failure programs to jointly transform the same preferred example. Applying the composition to the preferred representation gives

$$z^- = T_\pi(z^+),$$

which is rendered as $x^- = R(z^-)$. The resulting preference example is (c, x^+, x^-), where x^+ and x^- serve as the preferred and rejected images, respectively. Since z^- is obtained from the same editable source z^+, the preferred and rejected images differ through the deviations introduced by the composition of randomly sampled and instantiated failure programs. F-GPS therefore changes how the rejected image is constructed while remaining compatible with existing pairwise diffusion preference objectives. Additional details and representative examples are provided in Appendix F.

4 Experiments

We evaluate F-GPS on two T2I generation tasks: *image composition* and *visual text rendering*. These tasks provide complementary evaluations of structured alignment at different granularities: image composition tests object-level requirements such as object presence, attributes, spatial configuration, and scene context, whereas visual text rendering tests finer-grained character-level content, placement, and layout. Together, they allow us to evaluate whether the same failure-guided supervision framework remains effective across qualitatively different forms of structured generation failure.

4.1 TASK SETUP

(a) Image Composition. We use 15,335 samples from the `MuLAn` dataset (Tudosiu et al., 2024), where each scene is decomposed into composited RGBA layers, enabling controlled manipulation of object presence, spatial configuration, and visual properties. We augment this representation with object-level attributes, including position (`x/y` coordinates), scale, rotation, spatial layout, and color, together with object masks derived from non-zero alpha regions. These attributes expose editable visual factors that can be independently modified by failure programs; construction details are provided in the Appendix C. To support a broader range of programmatic transformations, we additionally incorporate foreground objects from the `MAGICK` dataset (Burgert et al., 2024), which provides generated objects with high-quality alpha mattes suitable for controlled insertion, and landscape backgrounds from Kaggle (datasets/arnaud58/landscape-pictures) for modifying environmental context. Image composition experiments are conducted on SD v1.5, SDXL-Base-1.0, and SD3.5-Medium.

(b) Visual Text Rendering. We use 6,218 samples from the `Crello` dataset (Yamaguchi, 2021), which provides structured graphic-design templates with background and text elements. Text-overlay images are rendered using `CyberAgentAILab/cr-renderer` (Inoue et al., 2024), which we extend from word-level annotations to character-level synthesis by deriving character-specific spatial and style attributes. This enables finer-grained control over character placement, rotation, spacing, and appearance; details of the character-level representation are provided in the Appendix C. Visual text rendering experiments are conducted on SDXL-Base-1.0.

Across both tasks, we instantiate the failure analyst $\mathcal{E}$ with `GPT-4o` (Achiam et al., 2023). A representative implementation of a failure program is shown in Figure 3. Additional details on the failure-program libraries, program parameters, seed prompts, analyst instructions, and implementation are provided in the Appendix.

4.2 TRAINING AND BASELINES

We evaluate F-GPS under three pairwise diffusion preference objectives: Diffusion-DPO (Wallace et al., 2024), Linear-DPO (Li et al., 2026), and DSPO (Zhu et al., 2025). Under each objective, we compare three strategies for constructing the rejected example in the preference pair. **MG (Model-Generated)** uses an output sampled from the target model under the same prompt, while **MG+RS (Model-Generated + Random Shuffling)** randomly reassigns these model-generated rejected examples across training examples, providing a non-targeted contrast control. **F-GPS** constructs rejected examples through failure-program application as described in Section 3. Within each backbone and preference objective, all three variants use the same prompts, preferred examples, and optimization configuration, differing only in the rejected examples used for training. All models are fine-tuned using LoRA; backbone-specific training hyperparameters are provided in the Appendix.

We additionally report the unaligned base model and standard supervised fine-tuning (SFT) as reference baselines. For SD v1.5 image composition, we compare against SPIN-Diffusion (Yuan et al., 2024), SePPO (Zhang et al., 2024), SPO (Liang et al., 2024), DRO (Wu et al., 2025), and RealAlign Chen et al. (2026). For SDXL-Base-1.0 image composition, we compare against SPO (Liang et al., 2024), InPO (Lu et al., 2025), and IterComp (Zhang et al., 2025). For visual text rendering with SDXL-Base-1.0, we compare against SPO (Liang et al., 2024) and InPO (Lu et al., 2025),

4.3 EVALUATION PROTOCOL

Task-Specific Evaluation. ***(a) Image Composition.*** We evaluate image composition on 750 prompts from each of T2I-CompBench++ (Huang et al., 2023) and TIFA (Hu et al., 2023). For T2I-CompBench++, we sample 75 prompts from each of its 10 compositional categories. We evaluate these prompts following the TIFA question-answering protocol (Hu et al., 2023): question–answer pairs are generated from each text prompt using `GPT-4o`, and image–text faithfulness is then measured by testing whether VQA models can recover the corresponding answers from the generated images. We use `microsoft/git-large` (Wang et al., 2022) and `Salesforce/blip-vqa-capfilt-large` (Li et al., 2022) as VQA evaluators.

(b) Visual Text Rendering. We evaluate visual text rendering on 500 prompts from each of the CVTG benchmark (Du et al., 2025) and MARIO-Eval (Chen et al., 2023). We report Exact Match

(EM), Normalized Exact Match (NEM), Partial String Similarity (PSS), Word Precision (WP), Word Recall (WR), and Word F1 (WF1). We use `llava-v1.5-13b` (Liu et al., 2024a) to extract text from the generated images using the prompt provided in Appendix K.

VLM-as-a-Judge Evaluation. In addition to task-specific evaluation, we use `Qwen3-VL-8B-Instruct` (Team, 2025) as a VLM judge to compare outputs produced by the MG, MG+RS, and F-GPS variants. For each evaluation prompt, the candidate images are jointly ranked with their presentation order randomized. We report the Top-1 win rate (%) in Figure 4, defined as the percentage of samples for which a variant is ranked first. We additionally report the normalized Borda score in the Appendix I, which summarizes the full ranking by assigning greater credit to higher-ranked variants and normalizing the aggregate score to a percentage scale.

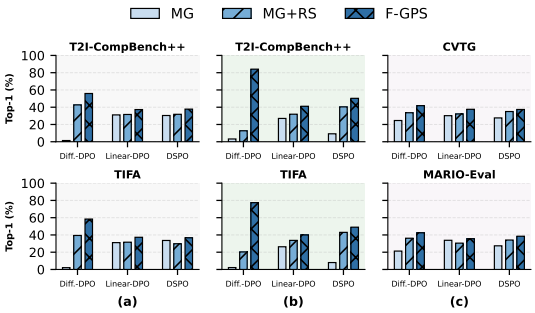


Figure 4: **VLM-as-a-Judge based evaluation.** Columns correspond to **(a)** SD v1.5 on image composition, **(b)** SDXL-Base-1.0 on image composition, and **(c)** SDXL-Base-1.0 on visual text rendering.

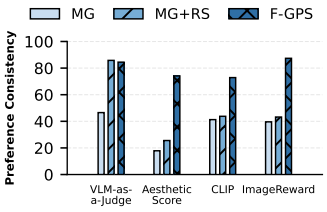
Image Quality. To assess whether improvements in structured alignment come at the cost of overall image quality, we report win rate (%) against the base model using Aesthetic Score (Schuhmann, 2022) and ImageReward (Xu et al., 2023) across both tasks.

5 Result and Discussion

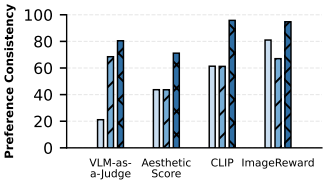
❶ **Does failure-guided preference supervision improve downstream alignment?** Across both tasks, F-GPS improves downstream alignment across different T2I backbones and preference objectives, as shown in Tables 1, 2, 3, with additional results reported in the Appendix I. Compared with the unaligned base model, F-GPS yields consistent gains; for example, on SD v1.5, F-GPS improves over the base model by 6.0 pp on T2I-CompBench++ (Avg.) and 6.2 pp on TIFA. It also generally performs better than alternative rejected-example constructions and existing alignment baselines. The VLM-as-a-Judge evaluation in Figure 4 provides complementary evidence: on SD v1.5, F-GPS improves the Top-1 win rate over MG+RS by 13.2 pp on T2I-CompBench++ and 19.0 pp on TIFA under Diffusion-DPO. Qualitative results shown in Figure 6 and Appendix further show that F-GPS more reliably satisfies the targeted compositional and text-rendering requirements, indicating that failure-guided construction of rejected examples improves structured alignment.

❷ **Why non-targeted rejected examples degrade alignment?** Preference optimization may underperform the base model when rejected examples provide a weak learning signal (Tables 1, 2, 3 and Appendix I). MG can substantially degrade structured alignment, while random shuffling in MG+RS often recovers part of this loss despite removing prompt-specific correspondence. For example, on SD3.5-Medium with Diffusion-DPO, the T2I-CompBench++ average drops from 85.8 for the base model to 48.9 with MG, recovers to 83.4 with MG+RS, and reaches 88.3 with F-GPS. These results highlight the importance of constructing rejected examples that provide a relevant preference contrast.

❸ **Does improved structured alignment preserve overall image quality?** We evaluate whether the gains from F-GPS come at the expense of broader image quality using Aesthetic Score and ImageReward as shown in Appendix I.5. Across most settings, F-GPS maintains or improves image quality, while alternative rejected-example constructions are less consistent. This is particularly evident on SD3.5-Medium, where MG severely degrades quality, reaching only 1.4% Aesthetic win rate and



(a) Image composition.



(b) Visual text rendering.

Figure 5: **Preference consistency of training pairs constructed by MG, MG+RS, and F-GPS.**

Table 1: **Quantitative comparison for SD v1.5 using blip-large as VQA evaluator. Bold** denotes the best result and underlining denotes the second-best result in each column across all configurations. "Avg." denotes the average over the ten T2I-CompBench++ categories.

Method	T2I-CompBench++											TIFA
	3D-Sp	Color	C-Act	C-Spa	Complex	Non-Sp	Num	Shape	Spatial	Texture	Avg.	
Base	65.3	70.3	76.0	77.5	76.5	82.5	61.6	63.6	76.9	69.9	72.0	83.0
SFT	66.2	71.8	75.0	75.8	76.4	84.1	65.1	63.6	74.8	70.7	72.4	82.6
SPIN-Diffusion	65.4	69.1	75.8	75.1	74.6	81.7	57.7	63.0	72.7	65.7	70.1	82.2
SePPO	69.3	73.3	76.3	77.1	80.3	84.1	63.9	65.0	77.2	72.8	73.9	84.6
SPO	65.2	73.7	75.2	76.2	75.0	78.3	58.7	62.9	74.2	70.5	71.0	80.9
DRO	70.6	74.3	78.8	78.2	77.5	84.6	64.3	66.1	79.7	72.0	74.6	86.1
RealAlign	71.3	72.5	76.2	76.5	78.6	84.2	60.7	64.2	76.6	73.0	73.4	83.9
Diffusion-DPO												
w/ MG	48.6	50.2	62.3	60.5	59.3	70.0	50.8	50.1	57.4	55.2	56.4	67.1
w/ MG+RS	71.5	**79.8**	76.5	76.8	82.3	84.0	62.3	66.3	77.9	73.8	75.1	86.2
w/ F-GPS	74.6	78.6	**79.0**	**79.5**	83.4	**85.9**	**68.4**	**68.7**	**85.8**	**76.3**	**78.0**	**89.2**
Linear-DPO												
w/ MG	66.0	72.5	74.2	77.0	75.7	83.4	59.2	61.7	78.6	71.2	72.0	84.4
w/ MG+RS	73.1	75.1	75.6	77.6	80.2	83.4	61.8	67.0	79.3	68.9	74.2	84.5
w/ F-GPS	**75.0**	77.2	78.7	79.1	**84.0**	84.0	63.5	67.8	82.9	74.3	76.7	88.4
DSPO												
w/ MG	65.2	70.6	75.0	75.2	75.1	80.3	61.1	65.7	78.9	69.1	71.6	83.5
w/ MG+RS	67.4	69.2	75.1	75.5	75.3	81.8	63.9	60.2	79.3	69.3	71.7	82.4
w/ F-GPS	72.8	75.0	77.8	78.3	82.2	84.6	65.6	68.0	83.5	71.5	75.9	87.1

Table 2: **Quantitative comparison for SDXL-Base using blip-large as VQA evaluator. Bold** denotes the best result and underlining denotes the second-best result in each column across all configurations. "Avg." denotes the average over the ten T2I-CompBench++ categories.

Method	T2I-CompBench++											TIFA
	3D-Sp	Color	C-Act	C-Spa	Complex	Non-Sp	Num	Shape	Spatial	Texture	Avg.	
Base	75.2	81.5	78.9	78.6	82.8	83.5	69.4	69.6	82.4	78.9	78.08	88.6
SFT	76.3	82.1	79.7	77.7	81.3	82.8	69.8	71.3	83.2	73.8	77.80	89.0
SPO	78.4	83.3	79.0	77.7	82.4	84.4	68.4	70.9	81.2	79.8	78.55	89.6
InPO	78.6	86.1	79.8	78.2	85.1	84.3	74.1	69.7	82.9	79.5	79.83	90.4
IterComp	76.1	82.4	77.8	78.8	81.5	83.9	71.2	68.8	82.2	74.5	77.7	89.3
Diffusion-DPO												
w/ MG	61.1	66.1	68.5	71.8	69.2	72.3	52.6	59.6	68.5	63.8	65.35	73.6
w/ MG+RS	70.5	80.2	73.3	74.0	76.0	77.1	59.5	63.2	75.4	69.2	71.84	83.1
w/ F-GPS	80.9	87.8	80.1	**81.4**	85.3	**87.0**	74.7	**74.1**	**85.6**	78.6	**81.55**	**92.3**
Linear-DPO												
w/ MG	78.6	82.6	77.9	76.8	81.0	84.6	69.5	70.6	77.6	77.2	77.64	88.2
w/ MG+RS	77.8	83.8	79.8	79.5	83.9	84.3	69.1	71.1	83.3	78.2	79.08	89.7
w/ F-GPS	81.4	**88.3**	**81.9**	79.5	**86.1**	86.6	71.5	72.3	84.7	**79.9**	81.22	**92.3**
DSPO												
w/ MG	72.9	74.3	74.6	75.3	76.8	79.5	66.0	67.8	76.8	71.3	73.53	84.4
w/ MG+RS	77.4	84.9	79.6	78.8	84.4	84.5	70.1	70.5	79.6	**79.9**	78.97	89.9
w/ F-GPS	**81.7**	85.6	79.6	81.1	85.1	86.6	**75.2**	73.9	84.6	77.7	81.11	92.2

even 0.0% under ImageReward on TIFA, whereas F-GPS retains 60.0% and 64.6%, respectively. These results indicate that the alignment gains from F-GPS are achieved without sacrificing overall generation quality.

❹ **How does the number of failure programs affect alignment performance?** We ablate the size of the failure-program library on SDXL-Base-1.0 for both tasks. Specifically, we construct F-GPS variants using 5, 10, and the full 17 failure programs while keeping the remaining training configuration unchanged. This analysis measures how the coverage of distilled failure structure affects downstream alignment. The corresponding results are reported in the Appendix I.3.

❺ **Does failure-guided construction produce more consistent preference pairs?** We compare the preference consistency of pairs constructed by MG, MG+RS, and F-GPS across both tasks, as shown in Figure 5. For each preferred–rejected pair (x^+, x^-), *VLM-as-a-Judge* measures whether the designated preferred image x^+ is favored over x^-. For ImageReward, Aesthetic Score, and CLIP similarity, consistency is measured as the percentage of pairs satisfying $s(x^+) > s(x^-)$ for scorer s. Across both tasks, F-GPS yields higher preference consistency in most cases, indicating more reliably ordered preference pairs.

❻ **Are the induced failure programs reliable and efficient to execute?** We evaluate the failure programs distilled from SDXL-Base-1.0 on both tasks by applying every program to 3,000 samples

Table 3: **Quantitative comparison for visual text rendering.** We report Exact Match (EM), Normalized Edit Match (NEM), Partial String Similarity (PSS), and word-level precision (WP), recall (WR), and F1 (WF1).

Method	CVTG						MARIO-Eval					
	EM	NEM	PSS	WP	WR	WF1	EM	NEM	PSS	WP	WR	WF1
Base	24.2	27.0	77.9	50.1	50.9	48.6	0.8	0.8	39.8	3.1	3.7	3.1
SFT	24.0	25.8	81.0	50.2	54.6	49.6	21.2	21.8	74.5	41.3	40.9	39.3
SPO	30.6	32.6	81.2	55.6	54.3	53.1	24.7	25.9	75.4	45.2	43.4	42.6
InPO	27.0	30.2	84.0	57.4	56.0	55.1	23.9	24.4	78.0	47.4	44.9	44.8
Diffusion-DPO												
w/ MG	20.2	23.6	79.4	51.8	51.0	49.5	20.0	21.4	73.4	42.4	41.2	40.4
w/ MG+RS	27.8	30.6	82.0	56.7	57.2	54.7	22.2	22.7	76.7	43.9	43.0	42.0
w/ F-GPS	32.0	32.8	81.3	58.2	**59.0**	56.8	24.9	26.1	76.8	48.1	46.7	45.8
Linear-DPO												
w/ MG	24.6	27.0	79.0	51.9	50.7	49.4	18.6	19.4	75.7	41.0	41.2	39.4
w/ MG+RS	26.8	30.2	81.4	54.7	55.3	52.9	21.4	21.6	75.9	42.8	42.7	41.3
w/ F-GPS	**34.4**	**34.8**	81.5	59.0	58.8	**57.4**	23.3	24.1	75.8	46.9	45.5	44.7
DSPO												
w/ MG	24.2	27.4	78.1	47.6	51.3	47.5	20.4	21.0	76.7	43.2	43.1	41.3
w/ MG+RS	26.2	29.0	**82.3**	56.8	56.4	54.1	21.6	21.8	76.1	44.3	42.8	42.0
w/ F-GPS	31.6	33.8	81.2	**59.0**	57.9	57.0	**25.7**	**26.9**	**78.1**	**48.6**	**48.0**	**46.7**

from the corresponding dataset. For each program execution, we record whether it succeeds and its execution time, and aggregate these statistics to obtain the execution success rate and average runtime. As reported in the Appendix I.4, the programs achieve high execution success across both tasks while requiring low per-program execution time.

❼ **Do F-GPS rejected examples preserve the failure patterns observed in the target model?** We randomly sample 100 F-GPS rejected examples, analyze their failure cues, and compare them with the 17 atomic visual cues distilled from target-model failures. As shown in Appendix I.6, we characterize semantic correspondence as *explicit* when a recovered cue directly captures the same failure, *implicit* when it reflects a broader, narrower, or related manifestation, and *unmatched* when no clear correspondence is found. We define *failure-cue coverage* as the fraction of target-model cues with at least one explicit or implicit correspondence. F-GPS recovers 16 of 17 cues, yielding 94.1% coverage and showing that F-GPS-constructed examples largely preserve the failure structure observed in the target model.

6 Conclusion

We introduced F-GPS, a failure-guided preference supervision framework that uses structured failures observed in a target T2I model to construct controlled preference contrasts. F-GPS atomizes sample-specific failures into reusable atomic visual cues, operationalizes them as parameterized failure programs, and applies sampled program compositions to editable preferred examples to generate targeted rejected images. Across image composition and visual text rendering, F-GPS improves downstream alignment across different diffusion backbones and preference objectives while largely preserving overall image quality, demonstrating that effective T2I preference alignment depends not only on how preferences are optimized, but also on how the preference contrast is constructed.



Figure 6: **Qualitative comparison of image composition results.** The top two rows correspond to outputs from SD v1.5, while the bottom two rows correspond to SDXL-Base-1.0.

AI USE STATEMENT

In this work, we used generative AI tools to assist with the generation and formatting of tables, language polishing and editing of manuscript text, and code generation for parts of the experimental and visualization pipeline. We also used generative AI tools to co-create components of the architecture diagram. All AI-generated code was reviewed, verified, and tested by the authors, and AI-assisted text, tables, and visual elements were manually checked before inclusion in the manuscript. The authors take full responsibility for the final content of this work, including all text, claims, results, code, tables, and figures produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed information to support reproduction of F-GPS and its evaluation. The main paper specifies the failure-discovery, atomization, failure-program construction, and application procedures, together with the training and evaluation setup. The Appendix further documents the dataset representations and preprocessing, failure-program libraries and parameterizations, preference-pair construction, training hyperparameters, evaluation metrics, additional results and ablations, failure-analyst instructions, seed prompts, and the prompt used for text extraction. Together, these materials describe the data construction, optimization, and evaluation procedures used in our experiments.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning. *arXiv preprint arXiv:2305.13301*, 2023.

Ryan D Burgert, Brian L Price, Jason Kuen, Yijun Li, and Michael S Ryoo. Magick: A large-scale captioned dataset from matting generated images using chroma keying. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22595–22604, 2024.

Jingye Chen, Yupan Huang, Tengchao Lv, Lei Cui, Qifeng Chen, and Furu Wei. Textdiffuser: Diffusion models as text painters. *Advances in Neural Information Processing Systems*, 36:9353–9387, 2023.

Weiyan Chen, Weijian Deng, Yao Xiao, Weijie Tu, ZiYi Dong, Ibrahim Radwan, Liang Lin, and Pengxu Wei. When preference labels fall short: Aligning diffusion models from real data. *arXiv preprint arXiv:2605.19839*, 2026.

Florinel-Alin Croitoru, Vlad Hondru, Radu Tudor Ionescu, Nicu Sebe, and Mubarak Shah. Curriculum direct preference optimization for diffusion and consistency models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 2824–2834, 2025.

Nikai Du, Zhennan Chen, Shan Gao, Zhizhou Chen, Xi Chen, Zhengkai Jiang, Jian Yang, and Ying Tai. Textcrafter: Accurately rendering multiple texts in complex visual scenes. *arXiv preprint arXiv:2503.23461*, 2025.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first international conference on machine learning*, 2024.

Ying Fan, Olivia Watkins, Yuqing Du, Hao Liu, Moonkyung Ryu, Craig Boutilier, Pieter Abbeel, Mohammad Ghavamzadeh, Kangwook Lee, and Kimin Lee. Reinforcement learning for fine-tuning text-to-image diffusion models. In *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS) 2023*. Neural Information Processing Systems Foundation, 2023.

Matt Gardner, Yoav Artzi, Victoria Basmov, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, et al. Evaluating models' local decision boundaries via contrast sets. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1307–1323, 2020.

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.

Ankush Gupta, Andrea Vedaldi, and Andrew Zisserman. Synthetic data for text localisation in natural images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2315–2324, 2016.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14953–14962. IEEE, 2023.

Kevin David Hayes, Micah Goldblum, Vikash Sehwag, Gowthami Somepalli, Ashwinee Panda, and Tom Goldstein. Finegrain: Evaluating failure modes of text-to-image models with vision language model judges. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

Yushi Hu, Benlin Liu, Jungo Kasai, Yizhong Wang, Mari Ostendorf, Ranjay Krishna, and Noah A Smith. Tifa: Accurate and interpretable text-to-image faithfulness evaluation with question answering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 20406–20417, 2023.

Yushi Hu, Otilia Stretcu, Chun-Ta Lu, Krishnamurthy Viswanathan, Kenji Hata, Enming Luo, Ranjay Krishna, and Ariel Fuxman. Visual program distillation: Distilling tools and programmatic reasoning into vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9590–9601, 2024.

Zijing Hu, Fengda Zhang, and Kun Kuang. D-fusion: Direct preference optimization for aligning diffusion models with visually consistent samples. *arXiv preprint arXiv:2505.22002*, 2025.

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.

Naoto Inoue, Kento Masui, Wataru Shimoda, and Kota Yamaguchi. Opencole: Towards reproducible automatic graphic design generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8131–8135, 2024.

Divyansh Kaushik, Eduard Hovy, and Zachary C Lipton. Learning the difference that makes a difference with counterfactually-augmented data. *arXiv preprint arXiv:1909.12434*, 2019.

Dohyun Kim, Seungwoo Lyu, Seung Wook Kim, and Paul Hongsuck Seo. Ddspo: Direct diffusion score preference optimization via stepwise contrastive policy-pair supervision. 2025.

Yuval Kirstain, Adam Polyak, Uriel Singer, Shahbuland Matiana, Joe Penna, and Omer Levy. Pick-a-pic: An open dataset of user preferences for text-to-image generation. *Advances in Neural Information Processing Systems*, 36:36652–36663, 2023.

Sanyam Lakhanpal, Shivang Chopra, Vinija Jain, Aman Chadha, and Man Luo. Refining text-to-image generation: Towards accurate training-free glyph-enhanced image generation. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 4372–4381. IEEE, 2025.

Kyungmin Lee, Xiahong Li, Qifei Wang, Junfeng He, Junjie Ke, Ming-Hsuan Yang, Irfan Essa, Jinwoo Shin, Feng Yang, and Yinxiao Li. Calibrated multi-preference optimization for aligning diffusion models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 18465–18475, 2025.

Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International conference on machine learning*, pp. 12888–12900. PMLR, 2022.

Kesong Li, Yixuan Xu, Kuo-kun Tseng, Weiyi Lu, Kan Liu, and Tao Lan. Linear-dpo: Linear direct preference optimization for diffusion and flow-matching generative models. *arXiv preprint arXiv:2605.21123*, 2026.

Shufan Li, Konstantinos Kallidromitis, Akash Gokul, Yusuke Kato, and Kazuki Kozuka. Aligning diffusion models by optimizing human utility. *arXiv preprint arXiv:2404.04465*, 2024.

Zhanhao Liang, Yuhui Yuan, Shuyang Gu, Bohan Chen, Tiankai Hang, Ji Li, and Liang Zheng. Step-aware preference optimization: Aligning preference with denoising performance at each step. *arXiv preprint arXiv:2406.04314*, 2(5):7, 2024.

Yiheng Lin, Siyu Jiao, Xiaohan Lan, Wei Zhou, Qi She, Fei Yu, Heyun Chen, Zhengwei Wang, Jinghuan Chen, Moran Li, et al. Textsculptor: Training and benchmarking scene text editing. *arXiv preprint arXiv:2605.21090*, 2026.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 26296–26306, 2024a.

Qihao Liu, Adam Kortylewski, Yutong Bai, Song Bai, and Alan Yuille. Discovering failure modes of text-guided diffusion models via adversarial search. In *Twelfth International Conference on Learning Representations*, pp. 1–25. OpenReview. net, 2024b.

Yunhong Lu, Qichao Wang, Hengyuan Cao, Xiaoyin Xu, and Min Zhang. Smoothed preference optimization via renoise inversion for aligning diffusion models with varied human preferences. In *Forty-second International Conference on Machine Learning*.

Yunhong Lu, Qichao Wang, Hengyuan Cao, Xierui Wang, Xiaoyin Xu, and Min Zhang. Inpo: Inversion preference optimization with reparametrized ddim for efficient diffusion model alignment. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 28629–28639, 2025.

Yoonjin Oh, Yongjin Kim, Hyomin Kim, Donghwan Chi, and Sungwoong Kim. Ospo: Object-centric self-improving preference optimization for text-to-image generation. *arXiv preprint arXiv:2506.02015*, 2025.

Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. Data programming: Creating large training sets, quickly. *Advances in neural information processing systems*, 29, 2016.

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*, pp. 10684–10695, 2022.

Som Sagar, Aditya Taparia, and Ransalu Senanayake. Llm-assisted red teaming of diffusion models through" failures are fated, but can be faded". *arXiv preprint arXiv:2410.16738*, 2024.

Christoph Schuhmann. LAION-Aesthetics. `https://laion.ai/blog/laion-aesthetics/`, 2022. Accessed: 2023-11-10.

Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11854–11864. IEEE, 2023.

Tomoyuki Suzuki, Kang-Jun Liu, Naoto Inoue, and Kota Yamaguchi. Layerd: Decomposing raster graphic designs into layers. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17783–17792. IEEE, 2025.

Qwen Team. Qwen3 technical report, 2025. URL `https://arxiv.org/abs/2505.09388`.

Petru-Daniel Tudosiu, Yongxin Yang, Shifeng Zhang, Fei Chen, Steven McDonagh, Gerasimos Lampouras, Ignacio Iacobacci, and Sarah Parisot. Mulan: A multi layer annotated dataset for controllable text-to-image generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22413–22422, 2024.

Sujith Vemishetty, Advitiya Arora, and Anupama Sharma. Towards evaluating robustness of prompt adherence in text to image models. *arXiv preprint arXiv:2507.08039*, 2025.

Bram Wallace, Meihua Dang, Rafael Rafailov, Linqi Zhou, Aaron Lou, Senthil Purushwalkam, Stefano Ermon, Caiming Xiong, Shafiq Joty, and Nikhil Naik. Diffusion model alignment using direct preference optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8228–8238, 2024.

Jianfeng Wang, Zhengyuan Yang, Xiaowei Hu, Linjie Li, Kevin Lin, Zhe Gan, Zicheng Liu, Ce Liu, and Lijuan Wang. Git: A generative image-to-text transformer for vision and language. 2022.

Xiaoshi Wu, Yiming Hao, Keqiang Sun, Yixiong Chen, Feng Zhu, Rui Zhao, and Hongsheng Li. Human preference score v2: A solid benchmark for evaluating human preferences of text-to-image synthesis. *CoRR*, 2023.

Yi-Lun Wu, Bo-Kai Ruan, Chiang Tseng, and Hong-Han Shuai. Ranking-based preference optimization for diffusion models from implicit user feedback. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Jiazheng Xu, Xiao Liu, Yuchen Wu, Yuxuan Tong, Qinkai Li, Ming Ding, Jie Tang, and Yuxiao Dong. Imagereward: learning and evaluating human preferences for text-to-image generation. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pp. 15903–15935, 2023.

Kota Yamaguchi. Canvasvae: Learning to generate vector graphic documents. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5481–5489, 2021.

Jinrui Yang, Qing Liu, Yijun Li, Soo Ye Kim, Daniil Pakhomov, Mengwei Ren, Jianming Zhang, Zhe Lin, Cihang Xie, and Yuyin Zhou. Generative image layer decomposition with visual effects. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7643–7653. IEEE, 2025.

Kai Yang, Jian Tao, Jiafei Lyu, Chunjiang Ge, Jiaxin Chen, Weihan Shen, Xiaolong Zhu, and Xiu Li. Using human feedback to fine-tune diffusion models without any reward model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8941–8951, 2024.

Shengming Yin, Zekai Zhang, Zecheng Tang, Kaiyuan Gao, Xiao Xu, Kun Yan, Jiahao Li, Yilei Chen, Yuxiang Chen, Heung-Yeung Shum, et al. Qwen-image-layered: Towards inherent editability via layer decomposition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16196–16205, 2026.

Huizhuo Yuan, Zixiang Chen, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning of diffusion models for text-to-image generation. *Advances in Neural Information Processing Systems*, 37: 73366–73398, 2024.

Daoan Zhang, Guangchen Lan, Dong-Jun Han, Wenlin Yao, Xiaoman Pan, Hongming Zhang, Mingxiao Li, Pengcheng Chen, Yu Dong, Christopher Brinton, et al. Seppo: Semi-policy preference optimization for diffusion alignment. *arXiv preprint arXiv:2410.05255*, 2024.

Xinchen Zhang, Ling Yang, Guohao Li, Yaqi Cai, Yong Tang, Yujiu Yang, Mengdi Wang, Bin CUI, et al. Itercomp: Iterative composition-aware feedback learning from model gallery for text-to-image generation. In *International Conference on Learning Representations*, volume 2025, pp. 31968–31988, 2025.

Huaisheng Zhu, Teng Xiao, and Vasant G Honavar. Dspo: Direct score preference optimization for diffusion model alignment. In *The Thirteenth International Conference on Learning Representations*, 2025.

Appendix Contents

A Limitations and Future Work **15**

B Related Work **15**

B.1 Failure Analysis in Text-to-Image Models 15

B.2 Programmatic Supervision and Controlled Data Construction 15

B.3 Preference Optimization for Diffusion Models 16

B.4 Preference Pair Construction for T2I Alignment 16

C Dataset Properties **16**

C.1 Image Composition 16

C.2 Visual Text Rendering 17

C.3 Prompt Generation 17

C.4 Representative Dataset Instances 17

D Failure Discovery and Atomization **20**

D.1 Identified Coarse Failure Instances 20

D.2 Atomization into Fine-Grained Visual Factors 20

E Failure-Program Libraries and Implementations **20**

F Failure Program Application **22**

G Implementation Details **29**

H Evaluation Metrics **29**

I Additional Results and Ablations **30**

I.1 Additional Evaluation with GIT-Large 30

I.2 Additional Experiments on SD3.5-Medium 30

I.3 Effect of the Number of Failure Programs 30

I.4 Failure-Program Execution Statistics 30

I.5 Image-Quality Evaluation 32

I.6 Failure-Cue Correspondence for F-GPS Rejected Examples 32

I.7 Qualitative Results 33

J Instructions Given to the Failure Analyst **38**

K Prompt Used for OCR-Based Evaluation **44**

L Prompts used for Failure analysis **44**

A Limitations and Future Work

Scope and fidelity of failure programs. F-GPS captures selected, operationalizable failures rather than the full range of failures exhibited by a target model. Program induction depends on the failure analyst and the available editable representation, and generated rejected examples may introduce unintended rendering artifacts. Although we evaluate preference pairs using automatic metrics and a VLM judge, more direct assessments of program fidelity could help distinguish intended failure cues from incidental visual changes.

Operationalizing failure knowledge. The induced program library provides a structured and executable representation of the failure patterns identified by the failure analyst. Rather than directly transferring the analyst's predictions or reasoning to the target model, F-GPS translates identified failure cues into reusable interventions for constructing preference supervision. Thus, the program library serves as a mechanism for encoding and applying failure knowledge during data construction, without implying any particular mechanism by which the target model acquires improved capabilities during fine-tuning.

Adaptive failure discovery and selection. In the current framework, failure programs and their parameters are sampled without explicitly optimizing their utility for preference training. Future work could adapt program selection and composition based on failure-specific evaluation, while maintaining diversity and fidelity to the intended failure structure. Repeating failure discovery at selected training checkpoints could also update the program library as the target model's observed failures change.

Extensions to other tasks and modalities. Editable RGBA decompositions at different levels of granularity (Tudosiu et al., 2024; Suzuki et al., 2025; Yin et al., 2026; Yang et al., 2025) offer a route to extending F-GPS to image editing, graphic design, and subject personalization. Another direction is physics-aware video generation, where failure programs could target temporal inconsistency and implausible object interactions. Such extensions would require representations that preserve the relevant spatial, identity, or physical relationships.

B Related Work

B.1 Failure Analysis in Text-to-Image Models

Despite rapid improvements in overall image quality, T2I models continue to exhibit diverse and structured generation failures. FineGRAIN (Hayes et al., 2025) organizes T2I behavior into a fine-grained taxonomy of failure modes spanning object attributes, counting, spatial relations, anatomy, physical relations, and text rendering. Compositional benchmarks such as T2I-CompBench (Huang et al., 2023), TIFA (Hu et al., 2023), and GenEval (Ghosh et al., 2023) expose persistent errors in attribute binding, object counting, spatial and non-spatial relations, object co-occurrence, and multi-object composition. Visual text generation presents a different set of challenges: TextDiffuser (Chen et al., 2023) highlights difficulties in rendering accurate and coherent text, while LenCom-Eval (Lakhanpal et al., 2025) further examines failures involving lengthy and complex text sequences. Recent evaluations further show that such prompt-adherence failures persist in newer T2I models, including for simple geometric and spatial requirements (Vemishetty et al., 2025). Complementary work moves beyond predefined benchmarks to actively discover failures: SAGE (Liu et al., 2024b) searches prompt and latent spaces for systematic undesirable behaviors, while LLM-assisted red teaming (Sagar et al., 2024) uses reinforcement-learning-based exploration together with LLM-generated rewards and states to characterize failure landscapes in diffusion models. Collectively, these studies show that T2I failures span diverse yet identifiable forms that can be systematically characterized. F-GPS builds on this observation by using failures observed in the target model not only for evaluation or diagnosis, but to guide the construction of preference supervision.

B.2 Programmatic Supervision and Controlled Data Construction

Programmatic supervision provides a way to encode reusable knowledge in executable programs rather than relying solely on instance-wise annotation. Data programming (Ratner et al., 2016)

formalizes domain heuristics as labeling functions that automatically generate weak supervision, while Kaushik et al. (2019) and Gardner et al. (2020) use controlled edits to isolate task-relevant distinctions in training and evaluation, respectively. In vision, systems such as VisProg (Gupta & Kembhavi, 2023) and ViperGPT (Surís et al., 2023) use LLM-generated programs to compose visual tools, while Visual Program Distillation (Hu et al., 2024) uses verified executable programs as an intermediate mechanism for synthesizing reasoning supervision for vision-language models (VLMs). Programmatic rendering has likewise been used to construct visual training data, from synthetic scene-text generation (Gupta et al., 2016) to recent large-scale text-rich data construction such as TextSculptor (Lin et al., 2026). F-GPS builds on this idea by *distilling recurring failure patterns observed in the target model into executable failure programs*. These programs operationalize the distilled failure knowledge as transformations that introduce targeted deviations into editable preferred examples, yielding rejected examples for preference alignment. The contribution therefore lies not in rendering or compositing itself, but in distilling observed model failures into reusable programs for constructing preference supervision. See Appendix B for extended related work.

B.3 Preference Optimization for Diffusion Models

Preference-based alignment for diffusion models builds on the broader RLHF paradigm, where human or model-derived preferences are used to steer generation toward preferred outputs (Black et al., 2023; Fan et al., 2023). More recent work has developed diffusion-specific preference optimization objectives that avoid or simplify the conventional reward-model-and-RL pipeline. Diffusion-DPO (Wallace et al., 2024) adapts direct preference optimization to the diffusion denoising process, while subsequent methods modify different components of the preference-learning formulation, including timestep-wise optimization, reference-policy treatment, and inversion-based training. Representative approaches include D3PO (Yang et al., 2024), Diffusion-KTO (Li et al., 2024), DSPO (Zhu et al., 2025), InPO (Lu et al., 2025), SmPO (Lu et al.), CaPO (Lee et al., 2025), DDSPO (Kim et al., 2025), and Linear-DPO (Li et al., 2026). Despite their differences, these methods primarily focus on how preference signals are translated into optimization objectives and model updates. F-GPS is complementary to this objective-design line: rather than introducing a new optimization rule, it focuses on how rejected examples used for preference alignment are constructed.

B.4 Preference Pair Construction for T2I Alignment

Early preference-based T2I alignment largely relied on human-annotated comparisons, with datasets such as Pick-a-Pic (Kirstain et al., 2023) and HPD v2 (Wu et al., 2023) collecting pairwise judgments over model-generated images. While such datasets provide direct human preference signals, collecting new comparisons at scale is costly, motivating increasing interest in constructing preference supervision automatically. SPIN-Diffusion (Yuan et al., 2024) iteratively derives supervision through competition between a model and its earlier versions, while SePPO (Zhang et al., 2024) leverages previous model checkpoints as reference policies to generate and selectively exploit reference samples. IterComp (Zhang et al., 2025) constructs composition-aware rankings over galleries generated by multiple diffusion models and uses them for iterative feedback learning. Other approaches construct more structured contrasts: D-Fusion (Hu et al., 2025) forms visually consistent preference samples through mask-guided fusion, while OSPO (Oh et al., 2025) constructs object-centric preference pairs for MLLM-based T2I generation through prompt perturbation, densification, and VQA-based scoring. Most closely related to our data-construction setting, RealAlign (Chen et al., 2026) treats curated real images as preferred references and constructs rejected counterparts through saliency-guided inpainting and filtering. F-GPS differs in deriving its rejected-example construction mechanism from failures observed in the target T2I model itself, operationalizing this failure knowledge as reusable parameterized transformations for constructing rejected examples with targeted deviations.

C Dataset Properties

C.1 Image Composition

We use 15,335 samples from the MuLAn dataset Tudosiu et al. (2024), where each scene is decomposed into a background and multiple composited RGBA object layers. This layered representation

enables controlled manipulation of object presence, spatial configuration, and visual properties. We augment it with object-level geometric and appearance attributes and derive object masks from non-zero alpha regions. The editable representation contains the following attribute groups:

Layer Attributes: `index`, `file`, `type`, `enabled`, `count`, `instances`.

Instance Attributes: `position`, `scale`, `rotation`, `flip_horizontal`, `flip_vertical`, `color_mode`, `color_intensity`, `saturation_boost`, `brightness_adjust`.

Geometry Attributes: `bbox`, `object_mask`, `canvas_size`.

Appearance Attributes: `original_image`, `image`.

Derived Features: `edge_mask`.

For interventions requiring additional visual assets, we incorporate foreground objects from MAGICK and landscape backgrounds from Kaggle to support object insertion and controlled changes to background context.

C.2 Visual Text Rendering

We use 6,218 samples from the Crello dataset Yamaguchi (2021), which provides design templates with background images and textual content as separate annotations. We render text-overlay images using `CyberAgentAILab/cr-renderer`, which we extend to support character-level synthesis by deriving spatial and style attributes from the provided word-level annotations. Our resulting editable representation contains 37 attributes organized into four categories. These describe the representation used by our pipeline, including derived attributes, rather than exclusively the original Crello annotations.

Canvas Attributes: `id`, `group`, `format`, `category`, `canvas_width`, `canvas_height`, `length`, `title`, `suitability`, `keywords`, `industries`, `preview`, `cluster_index`.

Element Attributes: `type`, `left`, `top`, `width`, `height`, `angle`, `opacity`, `color`, `image`.

Text Attributes: `text`, `font`, `font_size`, `font_bold`, `font_italic`, `text_line`, `text_color`, `text_align`, `capitalize`, `line_height`, `letter_spacing`, `x_offset`, `y_offset`, `rotation_offset`.

Additional Features: `prompt`.

C.3 Prompt Generation

For both tasks, we use `Qwen3-VL-4B-Instruct` (Team, 2025) to synthesize natural instruction-style prompts from the information available in each dataset sample. For image composition, prompts are generated from scene metadata describing object presence, attributes, counts, and spatial configuration. For visual text rendering, we condition prompt generation on the background image, target textual content, and associated layout metadata to obtain descriptions that specify both the visual context and the text to be rendered. The resulting prompts serve as the shared conditioning text for each editable preferred example and its programmatically generated rejected examples, ensuring that the preference pair differs through the applied failure program rather than the underlying instruction.

C.4 Representative Dataset Instances

To illustrate the structured annotations used by our framework, we present one representative instance from each dataset. Listing 1 shows an example annotation from the Crello dataset used for visual text rendering, while Listing 2 presents an example annotation from the MuLan dataset used for image composition.

```
{
    'id': '5b8567a21cc8aa5429280e84',
    'length': 7,
    'group': 3,
    'format': 17,
```

```
    'canvas_width': 300,
    'canvas_height': 250,
    'category': 20,
    'title': 'Teddy Bear in Baby Car Seat',

    'type': [2, 0, 0, 0, 1, 1, 2],

    'left': [
        -2.299999952316284,
        91.0,
        227.0,
        157.0,
        111.2580337524414,
        102.9981918334961,
        -0.4000000059604645
    ],

    'top': [
        -23.899999618530273,
        61.0,
        61.0,
        113.0,
        87.78488159179688,
        124.78646850585938,
        60.70000076293945
    ],

    'width': [
        304.29998779296875,
        138.0,
        3.0,
        6.0,
        97.0,
        114.0,
        92.80000305175781
    ],

    'height': [
        274.3999938964844,
        128.0,
        128.0,
        1.0,
        24.0,
        36.0,
        128.6999969482422
    ],

    'angle': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    'opacity': [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0],

    'text': [
        "",
        "",
        "",
        "",
        "Molly's Baby\nCar Seats",
        "You Baby Is\nSafe Now",
        ""
    ],

    'font': [0, 0, 0, 0, 14, 14, 0],
    'font_size': [0.0, 0.0, 0.0, 0.0, 10.0, 16.0],
    'text_align': [0, 0, 0, 0, 2, 2, 0],
    'capitalize': [False, False, False, False, False, False, False],
    'letter_spacing': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],

    'cluster_index': 17,

    'prompt':
    "Generate an image with a teddy bear in a baby car seat; "
    "overlay text: 'Molly's Baby\nCar Seats', "
    "'You Baby Is\nSafe Now'."
}
```


Listing 1: Sample from the Crello dataset.


```
{
    "image_id": "000000312582",
    "canvas_size": (612, 612),

```

```
    "layers": [

        {
            "index": 0,
            "type": "background",
            "enabled": True,
            "count": 1,

            "instance": {
                "position": (0, 0),
                "scale": 1.0,
                "rotation": 0.0,
                "flip_horizontal": False,
                "flip_vertical": False,
                "color_mode": None,
                "color_intensity": 0.5
            }
        },

        {
            "index": 1,
            "type": "instance",
            "enabled": True,
            "count": 1,

            "instance": {
                "position": (0, 0),
                "scale": 1.0,
                "rotation": 0.0,
                "flip_horizontal": False,
                "flip_vertical": False,
                "color_mode": None,
                "color_intensity": 0.5,
                "bbox": (221, 449, 239, 498)
            }
        },

        {
            "index": 2,
            "type": "instance",
            "enabled": True,
            "count": 1,

            "instance": {
                "position": (0, 0),
                "scale": 1.0,
                "rotation": 0.0,
                "flip_horizontal": False,
                "flip_vertical": False,
                "color_mode": None,
                "color_intensity": 0.5,
                "bbox": (242, 454, 269, 496)
            }
        },

        {
            "index": 3,
            "type": "instance",
            "enabled": True,
            "count": 1,

            "instance": {
                "position": (0, 0),
                "scale": 1.0,
                "rotation": 0.0,
                "flip_horizontal": False,
                "flip_vertical": False,
                "color_mode": None,
                "color_intensity": 0.5,
                "bbox": (207, 450, 277, 497)
            }
        }
    ]
}
```

Listing 2: Sample from the MuLan dataset.

D Failure Discovery and Atomization

D.1 Identified Coarse Failure Instances

The failure analyst $\mathcal{E}$ examines each target-model output together with its prompt to identify sample-specific coarse failure instances. Figures 7 and 8 present representative instances for image composition and visual text rendering, respectively. Each instance describes the error as observed in a particular sample and may encompass multiple visual deviations, which are subsequently decomposed into fine-grained visual factors through atomization.

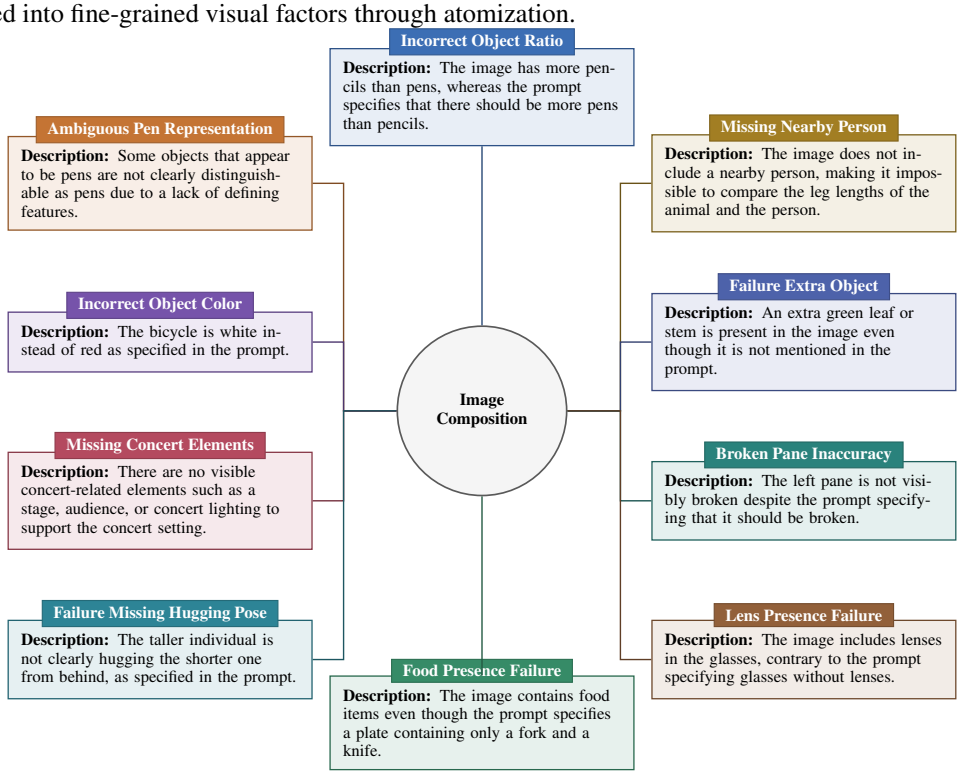


Figure 7: Representative coarse failure instances identified during the initial analysis of generated images for the **image composition** task.

D.2 Atomization into Fine-Grained Visual Factors

Following failure discovery, the failure analyst $\mathcal{E}$ atomizes each coarse failure instance into fine-grained visual factors that describe how the observed failure is manifested. Figures 9 and 10 illustrate representative decompositions for visual text rendering. The resulting factors form a candidate inventory that is subsequently filtered and aggregated across samples into reusable atomic visual cues.

E Failure-Program Libraries and Implementations

Using the retained atomic visual cues, we construct backbone-specific libraries of parameterized failure programs. For image composition, we obtain 17 programs for SD v1.5, 17 programs for SDXL-Base-1.0, and 16 programs for SD3.5-Medium. For visual text rendering, we obtain 17 programs for SDXL-Base-1.0. Each program operates on the corresponding editable representation and has a parameter domain controlling the applied intervention. Figures 11 and 12 summarize the complete failure-program libraries for image composition and visual text rendering on SDXL-Base-1.0, respectively, including the targeted atomic visual cues and key intervention parameters.

Visual Text Rendering. Figures 13 and 14 show representative failure-program implementations targeting small-text readability and irregular character spacing.

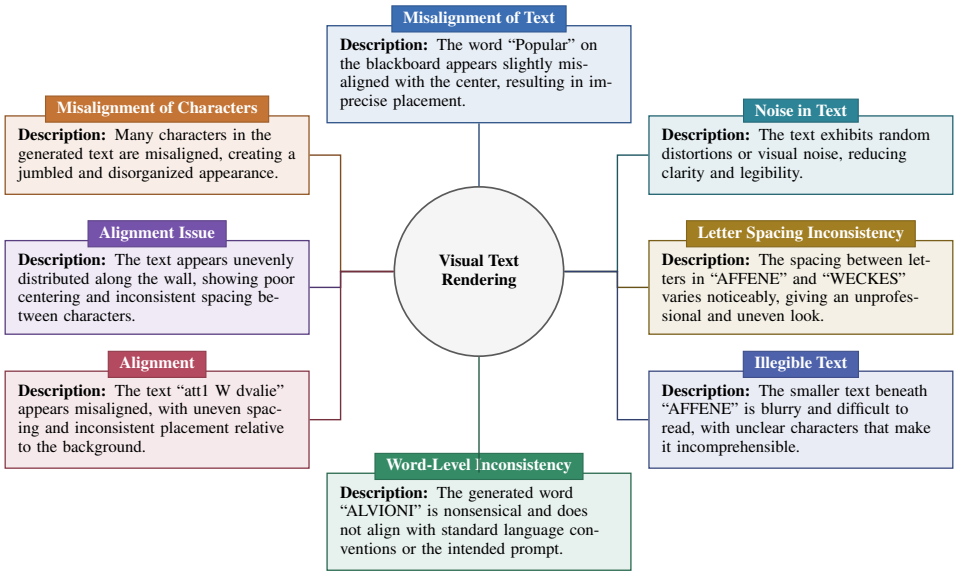


Figure 8: Representative coarse failure instances identified during the initial analysis of generated images for the **visual text rendering** task.

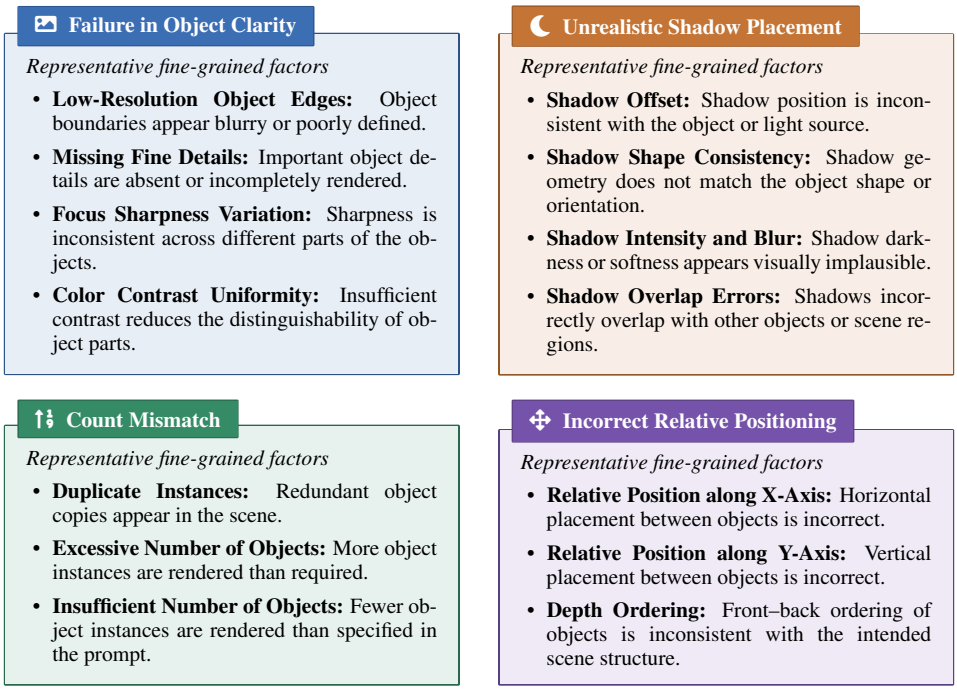


Figure 9: Representative examples of decomposing coarse failure modes in image composition into fine-grained, parameterizable visual factors that can subsequently be operationalized as visual programs..

Image Composition. Figures 15 and 16 show representative failure-program implementations that remove a required object or introduce additional object instances.

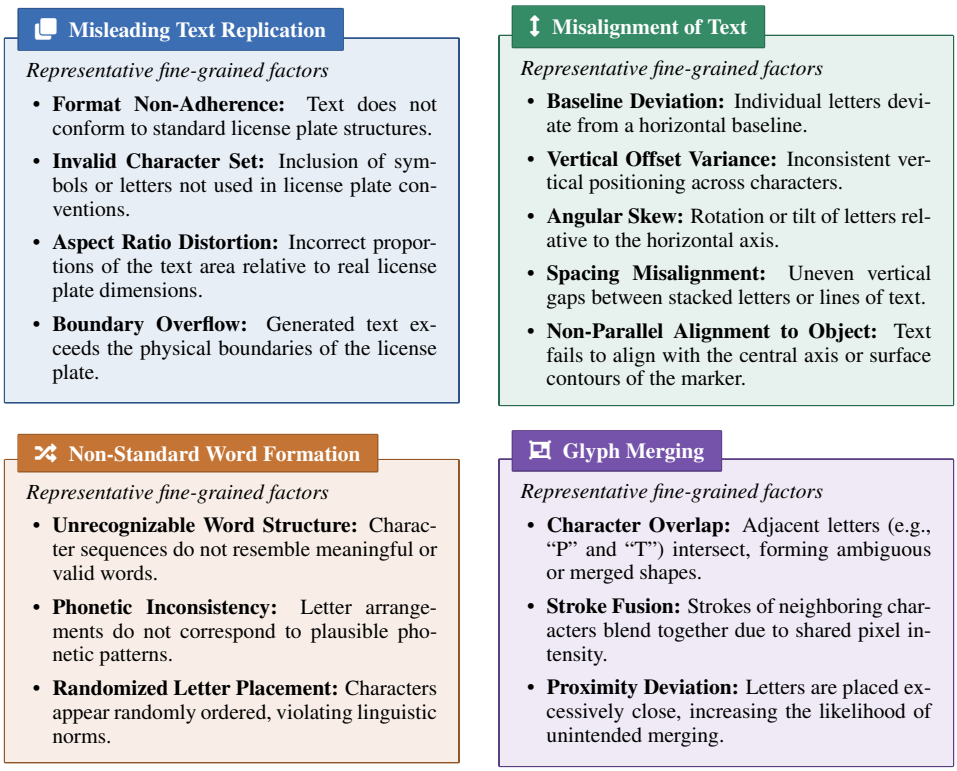


Figure 10: Representative examples of decomposing coarse failure modes into fine-grained, parameterizable visual factors that are implemented as visual programs.

F Failure Program Application

For each editable preferred example (c, z^+), we randomly sample a composition of failure programs from the corresponding task- and backbone-specific library, together with their application order and parameter values from the associated parameter domains. The instantiated programs are then applied sequentially to z^+, producing a modified representation z^- that is rendered into the rejected image $x^- = R(z^-)$. Together with the preferred image $x^+ = R(z^+)$, this yields the preference triplet (c, x^+, x^-), in which both images share the same conditioning prompt.

Representative Preference Pairs. Figure 17 presents F-GPS-generated preference pairs for image composition and visual text rendering. The examples illustrate how randomly sampled compositions of parameterized failure programs introduce targeted visual deviations into editable preferred examples to construct rejected images.

Comparison with Model-Generated Negatives. Figures 18 and 19 compare F-GPS-generated rejected images with samples generated directly by the base text-to-image model. For each prompt, we show the preferred image, the F-GPS-generated rejected image, and a model-generated sample. These comparisons illustrate the difference between constructing a rejected image through randomly sampled compositions of parameterized failure programs and obtaining one directly from the generative model.



Figure 11: Visual programs used for generating controlled image composition failures on the SDXL-Base-1.0 model. For each visual program, we report its resulting effect and key parameters.



Figure 12: Visual programs used for generating controlled visual text rendering failures on the SDXL-Base-1.0 model. For each visual program, we report its resulting effect and key parameters.

```python
def readability_deviation_in_small_text(example, transformation_parameters={"blur_intensity": 0.5, "pixelation_factor": 2, "contrast_reduction": 0.3, "spacing_reduction": 0.2}):

    """ Simulates readability deviation in small text."""
    # Create a copy to avoid modifying the original example
    new_example = copy.deepcopy(example)

    # Process all text elements
    for i, element_type in enumerate(new_example["type"]):
        if element_type == 1:

            # Reduce the effective font size to simulate pixelation
            original_font_size = new_example["font_size"][i]
            pixelated_font_size = max(1, original_font_size // transformation_parameters["pixelation_factor"])
            new_example["font_size"][i] = pixelated_font_size

            # Reduce text contrast by darkening/lightening RGB values
            text_color = new_example["text_color"][i]
            for j, color_value in enumerate(text_color):
                rgba_vals = list(map(float, color_value.lstrip("rgba(").rstrip(")").split(",")))
                rgba_vals[:3] = [val * (1 - transformation_parameters["contrast_reduction"]) for val in rgba_vals[:3]]
                text_color[j] = f"rgba({rgba_vals[0]}, {rgba_vals[1]}, {rgba_vals[2]}, {rgba_vals[3]})"

            new_example["text_color"][i] = text_color

            # Compress letter spacing to reduce readability
            original_spacing = new_example["letter_spacing"][i]
            reduced_spacing = original_spacing * (1 - transformation_parameters["spacing_reduction"])
            new_example["letter_spacing"][i] = reduced_spacing

    return new_example
```

Figure 13: Readability Deviation in Small Text Transformation

```python
def kerning_and_spacing_irregularity(example, transformation_parameters=None):
    """ Introduces uneven spacing between characters or words."""
    # Create a copy to preserve the original example
    new_example = copy.deepcopy(example)

    # Use default transformation parameters if none are provided
    transformation_parameters = transformation_parameters or {"max_offset": 5.0, "irregularity_probability": 0.5}

    # Iterate through all elements
    for i in range(new_example["length"]):
        if new_example["type"][i] == 1:

            # Determine the number of characters in the text
            text_length = len(new_example["text"][i])

            # Randomly perturb letter spacing for individual characters
            letter_spacing_values = [
                original_spacing + random.uniform(-transformation_parameters["max_offset"], transformation_parameters["max_offset"])
                if random.random() < transformation_parameters["irregularity_probability"] else original_spacing
                for original_spacing in [new_example["letter_spacing"][i]] * text_length
            ]

            # Store the average spacing value
            new_example["letter_spacing"][i] = sum(letter_spacing_values) / len(letter_spacing_values)

    return new_example
```

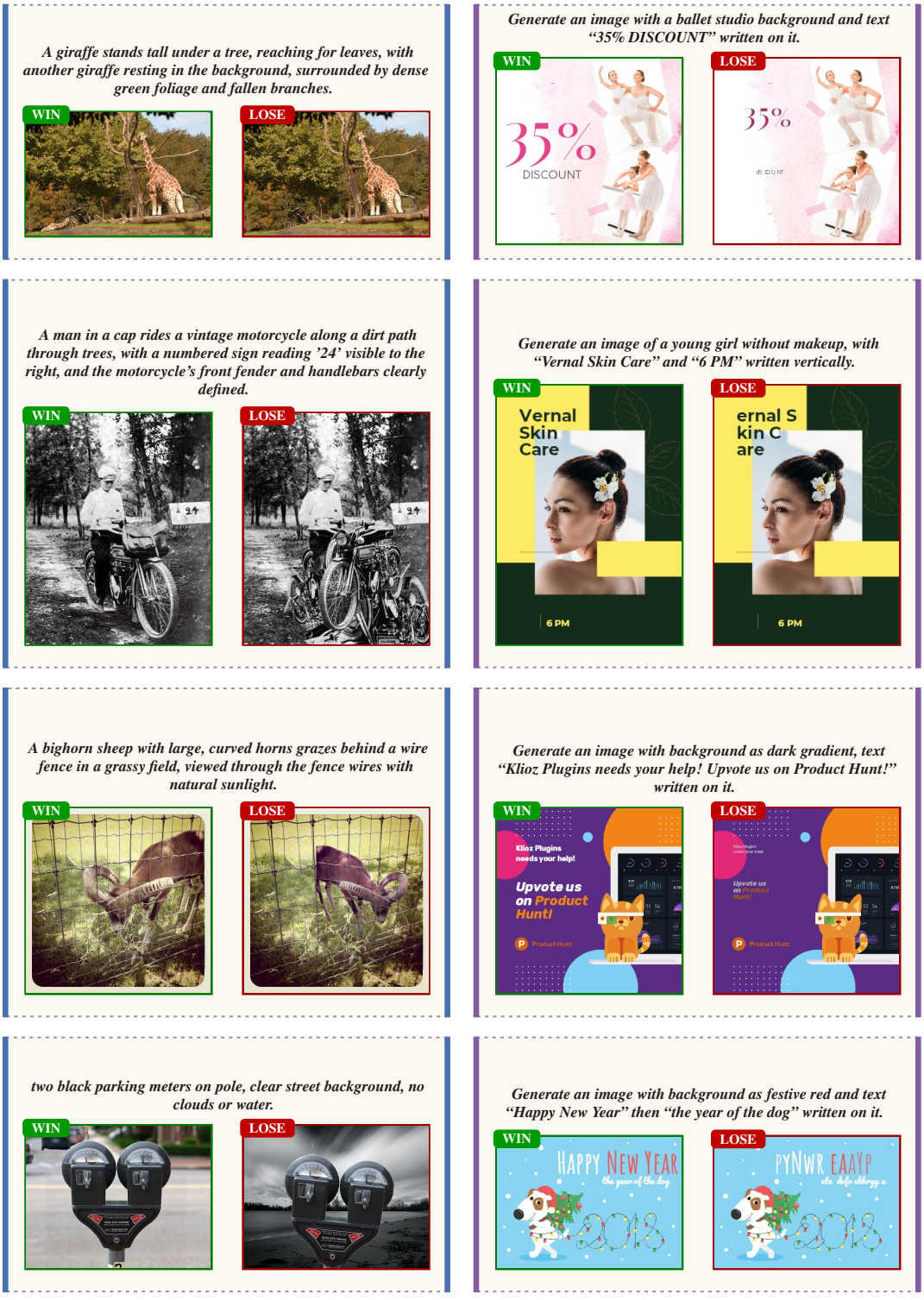
Figure 14: Kerning and Spacing Irregularity Transformation

```python
def missing_required_object(self, index: int = None, remove_random: bool = True):
        """ Removes an object layer to simulate the failure mode of a missing required object. """
        # Collect all valid instance layers
        valid_layers = [layer for layer in self.dataset.layers if layer["type"] == "instance" and layer["enabled"]]

        # Handle edge case: No valid layers to disable
        if not valid_layers:
            return  # No operation if no valid layers to remove

        if remove_random:
            # Randomly select a valid layer to disable
            layer_to_remove = random.choice(valid_layers)
        else:
            # Validate index and retrieve the specified layer
            if not (0 <= index < len(self.dataset.layers)):
                return  # Invalid index
            layer_to_remove = self.dataset.get_layer(index)
            if not layer_to_remove or layer_to_remove["type"] == "background" or not layer_to_remove["enabled"]:
                return  # Invalid layer for removal

        # Disable the selected layer
        layer_to_remove["enabled"] = False
```

Figure 15: Missing Required Object Transformation

```python
def incorrect_object_count(self, index: int, extra_count: int = 1, replace_original: bool = False):
        """ Failure mode where additional instances of an object are added, resulting in an incorrect object count.
        """
        target_layer = self.dataset.layers[index]

        # Check for critical fields and handle None values
        inst = target_layer['instances'][0]
        if inst.get('image') is None or inst.get('bbox') is None:
            return

        # Hide the original layer if specified
        if replace_original:
            target_layer['enabled'] = False

        # Get object's bounding box dimensions for dynamic randomization
        x1, y1, x2, y2 = inst['bbox']
        bbox_width = x2 - x1
        bbox_height = y2 - y1

        for _ in range(extra_count):
            new_layer = copy.deepcopy(target_layer)
            new_layer['index'] = len(self.dataset.layers)

            # Randomize position based on bbox size
            inst_new = new_layer['instances'][0]
            inst_new['position'] = (inst_new['position'][0] + random.randint(-bbox_width // 2, bbox_width // 2),
                inst_new['position'][1] + random.randint(-bbox_height // 2, bbox_height // 2))

            inst_new['rotation'] = random.uniform(-15, 15)
            inst_new['scale'] = inst_new['scale'] * random.uniform(0.8, 1.2)
            inst_new['flip_horizontal'] = random.choice([True, False])
            inst_new['flip_vertical'] = random.choice([True, False])
            inst_new['brightness_adjust'] = random.uniform(-0.2, 0.2)
            inst_new['saturation_boost'] = random.uniform(-0.2, 0.2)

            # Append the new layer to the dataset
            self.dataset.layers.append(new_layer)
```

Figure 16: Incorrect Object Count Transformation



(a) **Image composition** (b) **Visual text rendering**

Figure 17: **Examples of F-GPS-generated preference pairs. Left:** Image composition. **Right:** Visual text rendering. For each prompt, the winning image (WIN) satisfies the desired semantics, while the losing image (LOSE) is constructed by applying a randomly sampled composition of parameterized failure programs that introduce targeted visual deviations.

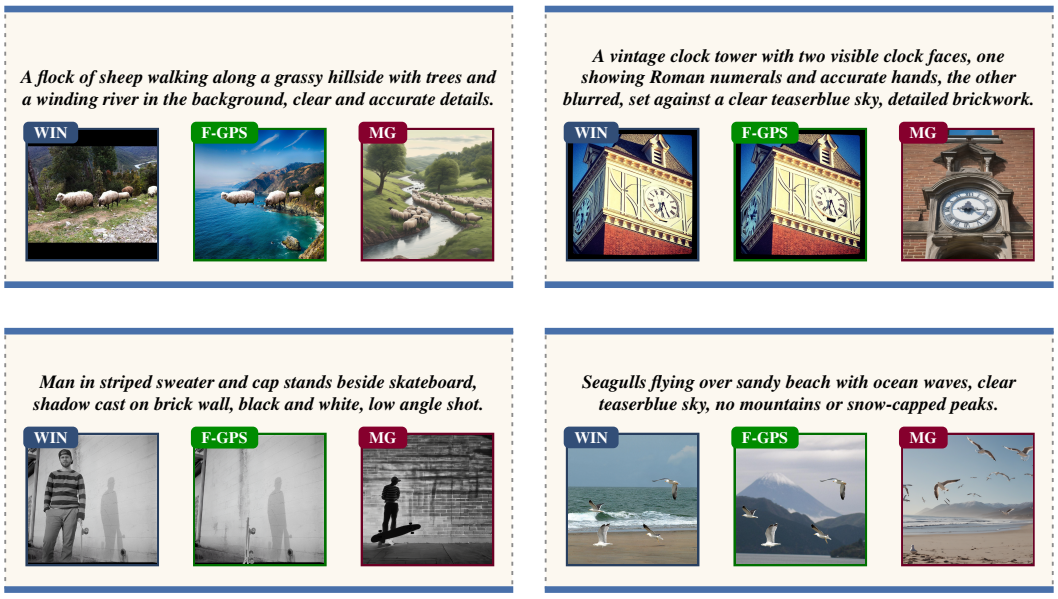


Figure 18: **Comparison of F-GPS and model-generated samples for image composition.** For each prompt, we show the reference winning image (WIN), the F-GPS losing image (F-GPS) constructed using a randomly sampled composition of parameterized failure programs, and a sample generated directly by the target model (MG).



Figure 19: **Comparison of F-GPS and model-generated samples for visual text rendering.** For each prompt, we show the reference winning image (WIN), the F-GPS losing image (F-GPS) constructed using a randomly sampled composition of parameterized failure programs, and a sample generated directly by the target model (MG).

G Implementation Details

All experiments are conducted on a Linux-based system equipped with a single NVIDIA GPU with **32 GB** of GPU memory. Training hyperparameters are reported in Tables 4 and 5.

Hyperparameter	Value
Base model	Stable Diffusion v1.5
Image resolution	512×512
Batch size	16
Gradient accumulation steps	2
Gradient checkpointing	Enabled
Optimizer	8-bit Adam
LoRA rank	128
Learning rate	5×10^{-5}
Learning rate scheduler	Constant
Warmup steps	0
Training steps	1000
Checkpoint interval	Every 50 steps
η_{DPO}	0.001
DMPO α	0.01
Random seed	0

Table 4: Training hyperparameters for SD v1.5.

Hyperparameter	Value
Base model	Stable Diffusion XL Base 1.0
VAE	madebyollin/sdxl-vae-fp16-fix
Mixed precision	FP16
Image resolution	512×512
Batch size	16
Gradient accumulation steps	2
Gradient checkpointing	Enabled
Optimizer	8-bit Adam
LoRA rank	128
Learning rate	5×10^{-5}
Learning rate scheduler	Constant
Warmup steps	0
Training steps	500
Checkpoint interval	Every 50 steps
η_{DPO}	0.001
DMPO α	0.01
Random seed	0

Table 5: Training hyperparameters for SDXL-Base.

H Evaluation Metrics

Table 6 summarizes the evaluation metrics used in our experiments, along with their descriptions and desired directions.

Metric	Description	Desired
EM	Exact Match. Measures the percentage of predictions whose recognized text exactly matches the ground-truth text.	↑
NEM	Normalized Exact Match. Computes exact match after text normalization, making the metric less sensitive to formatting variations.	↑
PSS	Position-aware String Similarity. Measures character-level similarity while accounting for character positions, providing partial credit for near-correct predictions.	↑
WP	Word Precision. Measures the proportion of predicted words that correctly match the ground-truth words.	↑
WR	Word Recall. Measures the proportion of ground-truth words successfully recovered in the prediction.	↑
WF1	Word-level F1 Score. Harmonic mean of Word Precision and Word Recall, providing a balanced measure of word-level recognition performance.	↑
TIFA	Text-to-Image Faithfulness Assessment. A VQA-based metric that evaluates whether the generated image faithfully satisfies the semantics specified by the text prompt.	↑
Aesthetic Score	Measures the visual appeal and overall aesthetic quality of generated images using a learned aesthetic predictor.	↑
ImageReward	A learned reward model that estimates human preference by measuring the alignment between a generated image and its corresponding text prompt.	↑

Table 6: Evaluation metrics used in the evaluation.

I Additional Results and Ablations

I.1 Additional Evaluation with GIT-Large

For SD v1.5 and SDXL-Base-1.0, we provide additional image-composition evaluation using `git-large` as the VQA evaluator. Tables 8 and 9 report the corresponding results across preference objectives and rejected-example construction strategies. The `git-large` results show a similar trend to the `blip-large`-based evaluation reported in the main paper, with F-GPS generally providing stronger alignment than MG and MG+RS across both backbones. We also report normalized Borda scores for the VLM-as-a-Judge evaluation in Figure 20, complementing the Top-1 win rates reported in the main paper. The Borda results exhibit a similar overall trend, with F-GPS receiving stronger preferences across the evaluated settings.

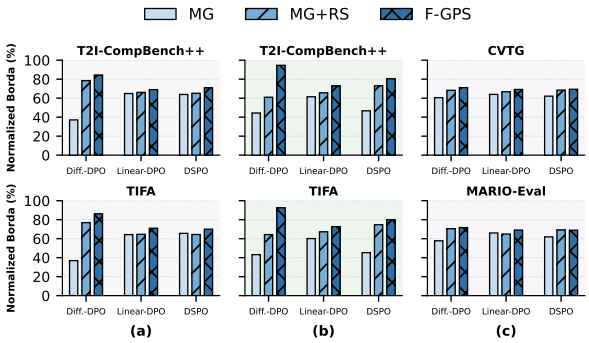


Figure 20: **VLM-as-a-Judge comparison using normalized Borda score.** Columns correspond to **(a)** SD v1.5 on image composition, **(b)** SDXL-Base-1.0 on image composition, and **(c)** SDXL-Base-1.0 on visual text rendering.

I.2 Additional Experiments on SD3.5-Medium

We further evaluate F-GPS on SD3.5-Medium for image composition using Diffusion-DPO. Following the same evaluation protocol, we train with a subset of 4,500 preference pairs and compare MG, MG+RS, and F-GPS. As shown in Table 7, F-GPS receives substantially stronger VLM-as-a-Judge preference under both Top-1 win rate and normalized Borda score on T2I-CompBench++ and TIFA. In contrast, MG yields very weak Top-1 preference, while random shuffling in MG+RS recovers part of the performance but remains well below F-GPS. These results further support the importance of constructing rejected examples with a targeted and controlled preference contrast. We additionally evaluate the resulting models with two VQA evaluators, `blip-large` and `git-large`, reported in Tables 10 and 11, respectively. These evaluations provide complementary evidence at the benchmark-metric level and show that the advantage of F-GPS is not specific to a single VLM-based comparison protocol.

Method	T2I-CompBench++		TIFA	
	Top-1	NB	Top-1	NB
Diff.-DPO				
w/ MG	0.0	36.4	0.0	33.4
w/ MG+RS	5.8	65.5	9.9	69.9
w/ F-GPS	**94.2**	**98.1**	**90.1**	**96.7**

Table 7: **VLM-as-a-Judge evaluation for SD3.5-Medium.** Results are reported on T2I-CompBench++ and TIFA using Top-1 win rate and normalized Borda (NB) score.

I.3 Effect of the Number of Failure Programs

We examine the effect of failure-program library size by constructing preference data using 5, 10, and 17 failure programs. For image composition, we average performance across T2I-CompBench++ and TIFA within each VQA evaluator; for visual text rendering, we average the corresponding metric values across CVTG and MARIO-Eval. This aggregation provides a clearer view of the overall effect of library size across datasets.

I.4 Failure-Program Execution Statistics

Figure 23 reports the execution statistics of the failure-program libraries distilled from SDXL-Base-1.0 for image composition and visual text rendering. For each task, we apply every induced failure program to 3,000 samples from the corresponding dataset and record whether the program executes successfully together with its execution time. We aggregate these runs to report the execution success rate and average runtime for each program. Across both tasks, the programs exhibit high

Table 8: **Quantitative comparison for SD v1.5 using git-large as VQA evaluator. Bold** denotes the best result and underlining denotes the second-best result in each column across all configurations. "Avg." denotes the average over the ten T2I-CompBench++ categories.

Method	T2I-CompBench++											TIFA
	3D-Sp	Color	C-Act	C-Spa	Complex	Non-Sp	Num	Shape	Spatial	Texture	Avg.	
Base	66.9	73.5	70.2	70.4	75.0	79.3	58.2	59.3	71.4	64.0	68.8	81.1
SFT	66.2	71.3	70.6	69.5	73.0	80.5	57.3	61.0	71.8	66.8	68.8	80.7
SPIN-Diffusion	70.6	70.5	70.3	70.6	74.2	80.9	50.3	59.3	71.6	62.7	68.1	81.1
SePPO	73.4	73.0	71.8	72.4	77.6	81.9	60.6	63.5	75.9	66.9	71.7	83.3
SPO	70.7	73.9	67.8	69.8	73.6	76.0	56.5	59.0	69.0	66.0	68.2	78.2
DRO	73.6	75.6	71.8	70.8	76.8	**83.7**	61.8	64.9	77.6	71.8	72.8	84.5
RealAlign	71.2	74.9	72.6	70.2	74.4	81.8	57.6	64.2	74.1	66.6	70.77	82.3
Diffusion-DPO												
w/ MG	52.4	51.6	54.9	55.9	55.8	64.7	46.3	48.1	56.5	53.1	53.9	64.6
w/ MG+RS	75.4	77.2	69.5	72.6	79.2	80.3	61.9	63.3	76.8	72.5	72.9	84.0
w/ F-GPS	76.4	**80.0**	72.4	72.4	**81.0**	80.5	**63.9**	65.3	**83.1**	73.0	**74.8**	**88.0**
Linear-DPO												
w/ MG	71.0	76.1	70.6	72.4	73.9	79.0	56.1	58.3	73.3	67.2	69.8	82.5
w/ MG+RS	75.0	76.7	73.0	72.3	77.7	80.2	60.0	63.0	75.6	66.7	72.0	83.3
w/ F-GPS	**77.1**	78.2	74.3	73.5	80.7	82.5	62.9	63.4	80.5	70.0	74.3	87.2
DSPO												
w/ MG	67.7	71.0	69.7	71.6	73.3	77.2	57.5	61.0	72.9	69.5	69.1	80.5
w/ MG+RS	71.0	69.9	71.0	71.4	74.4	77.8	60.6	58.8	75.3	65.1	69.5	80.9
w/ F-GPS	74.0	76.0	**74.8**	**75.1**	77.4	82.1	63.4	**65.5**	80.6	**73.2**	74.2	85.6

Table 9: **Quantitative comparison for SDXL-Base using `git-large` as VQA evaluator. Bold** denotes the best result and underlining denotes the second-best result in each column across all configurations. "Avg." denotes the average over the ten T2I-CompBench++ categories.

Method	T2I-CompBench++											TIFA
	3D-Sp	Color	C-Act	C-Spa	Complex	Non-Sp	Num	Shape	Spatial	Texture	Avg.	
Base	80.4	81.7	74.1	74.8	77.7	80.5	65.4	62.9	80.5	75.3	75.3	87.2
SFT	78.1	81.7	74.1	73.5	80.2	80.9	66.4	62.7	78.4	70.2	74.6	87.1
SPO	79.5	83.0	72.4	73.5	81.0	82.9	68.0	63.7	83.4	**77.2**	76.5	87.3
InPO	81.4	83.4	75.0	76.7	80.7	81.7	71.1	64.9	81.8	75.0	77.2	88.1
IterComp	81.1	83.4	74.1	74.6	81.2	81.5	65.6	60.6	79.4	73.2	72.1	87.1
Diffusion-DPO												
w/ MG	60.3	63.9	58.1	62.4	58.5	66.2	47.7	51.8	63.1	56.0	58.8	66.3
w/ MG+RS	70.5	79.0	68.9	67.4	71.6	75.9	58.4	58.0	71.7	66.6	68.8	79.5
w/ F-GPS	85.6	**86.2**	**76.6**	**77.5**	**83.5**	83.6	72.6	67.0	**85.2**	75.1	**79.3**	**90.8**
Linear-DPO												
w/ MG	82.1	82.6	73.4	73.6	77.3	81.5	64.2	62.1	76.5	74.6	74.8	87.0
w/ MG+RS	85.3	80.6	76.5	76.4	78.0	82.7	66.0	64.5	81.2	76.3	76.8	87.5
w/ F-GPS	83.8	86.0	76.3	**77.5**	80.8	83.3	70.5	67.3	81.4	**77.2**	78.4	90.5
DSPO												
w/ MG	77.5	75.2	72.0	68.6	74.1	76.0	60.6	61.1	78.4	68.8	71.2	83.5
w/ MG+RS	81.8	81.9	76.3	74.2	80.9	83.0	68.2	64.4	79.8	75.4	76.6	87.5
w/ F-GPS	**85.9**	85.3	75.6	75.4	82.9	**84.1**	**73.0**	**68.0**	83.0	76.4	79.0	90.3

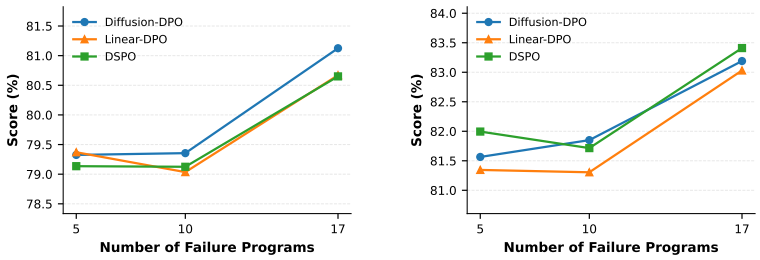
Table 10: **Quantitative comparison for SD3.5-Medium using `blip-large` as VQA evaluator. Bold** denotes the best result and underlining denotes the second-best result in each column across all configurations. "Avg." denotes the average over the ten T2I-CompBench++ categories.

Method	T2I-CompBench++											TIFA
	3D-Sp	Color	C-Act	C-Spa	Complex	Non-Sp	Num	Shape	Spatial	Texture	Avg.	
Base	84.0	95.0	**84.0**	93.0	85.0	90.0	81.0	78.0	82.0	86.0	85.8	95.0
Diffusion-DPO												
w/ MG	36.0	35.0	53.0	52.0	48.0	42.0	16.0	38.0	86.0	83.0	48.9	36.0
w/ MG+RS	81.0	90.0	80.0	92.0	80.0	88.0	77.0	77.0	86.0	83.0	83.4	93.0
w/ F-GPS	**89.0**	**96.0**	83.0	**94.0**	**88.0**	**91.0**	**83.0**	**82.0**	**89.0**	**88.0**	**88.3**	**96.0**

execution success rates and low per-program runtime, indicating that the distilled transformations can be applied reliably and efficiently during rejected-example construction.

Table 11: **Quantitative comparison for SD3.5-Medium using `git-large` as VQA evaluator. Bold** denotes the best result and underlining denotes the second-best result in each column across all configurations. "Avg." denotes the average over the ten T2I-CompBench++ categories.

Method	T2I-CompBench++											TIFA
	3D-Sp	Color	C-Act	C-Spa	Complex	Non-Sp	Num	Shape	Spatial	Texture	Avg.	
Base	85.0	**93.0**	**80.0**	87.0	81.0	86.0	77.0	68.0	87.0	82.0	82.6	**93.0**
Diffusion-DPO												
w/ MG	29.0	25.0	29.0	19.0	22.0	23.0	10.0	18.0	82.0	76.0	33.3	19.0
w/ MG+RS	80.0	85.0	72.0	79.0	74.0	83.0	67.0	70.0	82.0	76.0	76.8	88.0
w/ F-GPS	**90.0**	**93.0**	79.0	**90.0**	**83.0**	**88.0**	**82.0**	**76.0**	**93.0**	**87.0**	**86.1**	**93.0**



(a) GIT-Large **(b) BLIP-Large**

Figure 21: **Effect of the number of failure programs on image composition.** We vary the failure-program library size using 5, 10, and 17 programs and report performance under Diff.-DPO, Linear-DPO, and DSPO. Results are shown using (a) GIT-Large and (b) BLIP-Large as VQA evaluators.

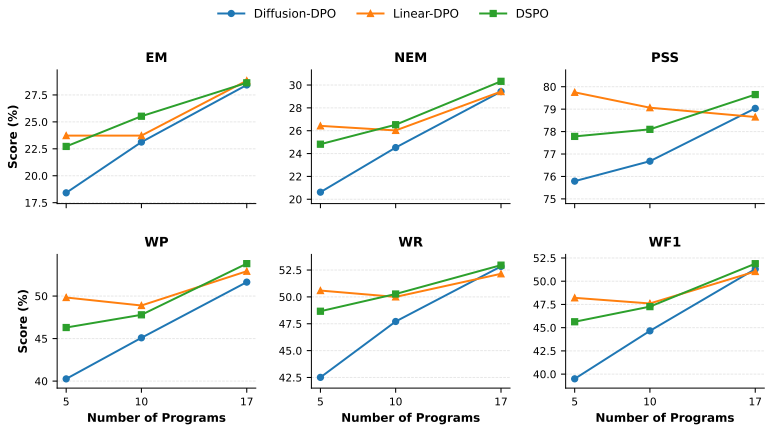


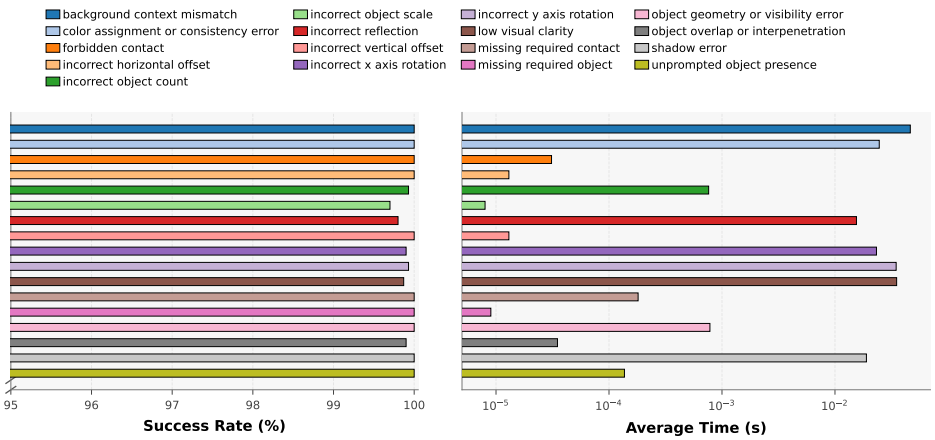
Figure 22: **Effect of the number of failure programs on visual text rendering.** We vary the failure-program library size using 5, 10, and 17 programs and report performance under Diffusion-DPO, Linear-DPO, and DSPO across EM, NEM, PSS, WP, WR, and WF1.

I.5 Image-Quality Evaluation

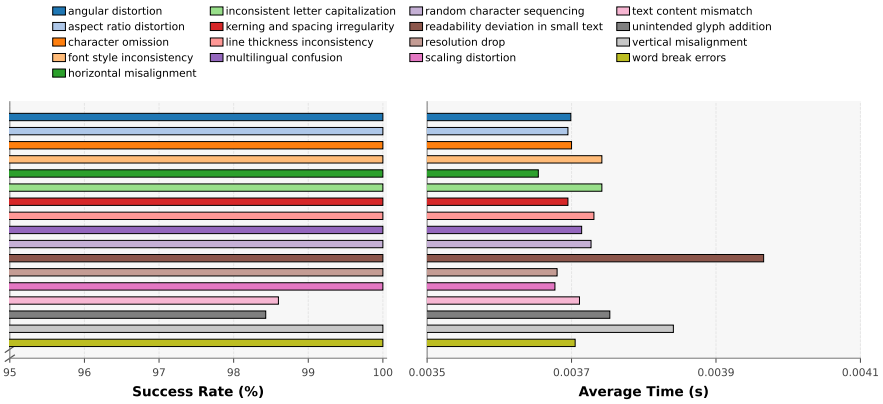
We evaluate whether improvements in structured generation are accompanied by changes in image quality using Aesthetic Score and ImageReward. Figures 24, 25, and 26 present the corresponding results for image composition and visual text rendering.

I.6 Failure-Cue Correspondence for F-GPS Rejected Examples

Figure 27 provides the detailed correspondence between the 17 atomic visual cues distilled from target-model failures and the failure cues independently recovered from randomly sampled F-GPS rejected examples. Direct, refined, and unmatched correspondences are shown separately.



(a) Image Composition

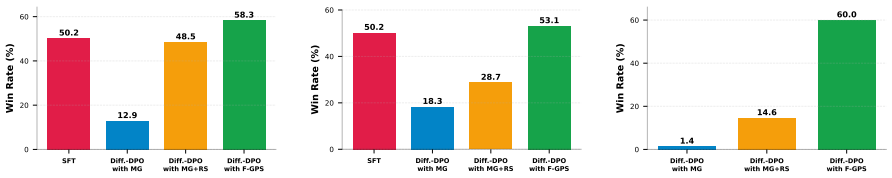


(b) Visual Text Rendering

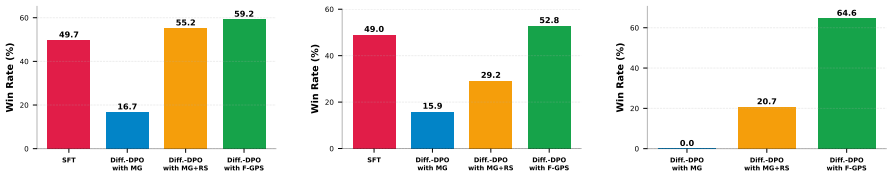
Figure 23: **Execution reliability and efficiency of the failure-program libraries distilled from SDXL-Base-1.0.**

I.7 Qualitative Results

We present additional qualitative comparisons across different preference optimization algorithms and diffusion backbones in Figures 28, 29, and 30.

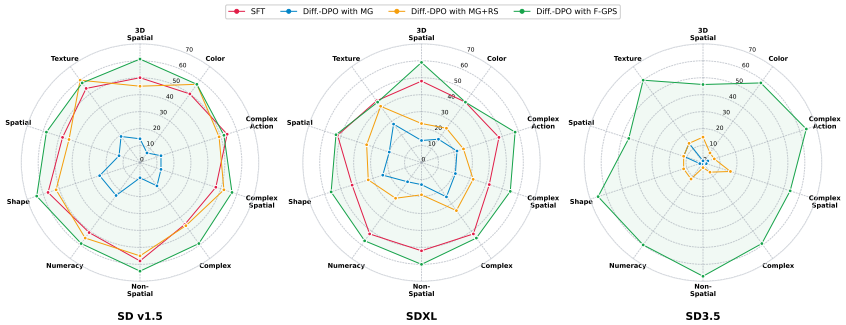


(a) Aesthetic Score.

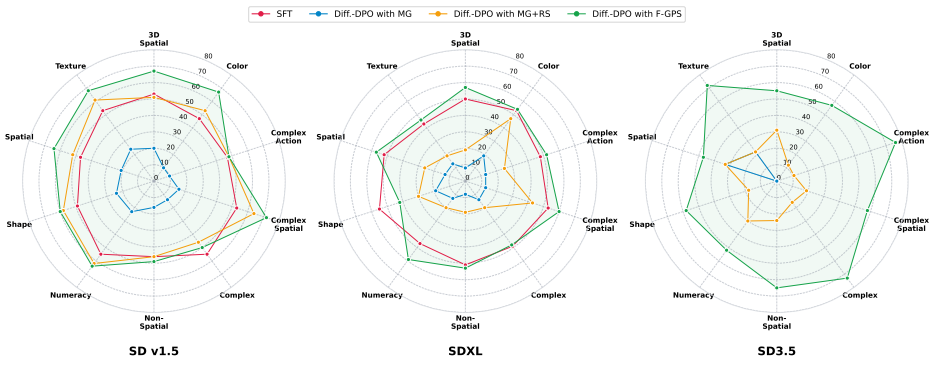


(b) ImageReward.

Figure 24: **Quality evaluation for image composition on the TIFA benchmark.** Results are shown for SD v1.5, SDXL-Base-1.0, and SD3.5-Medium from left to right. We compare SFT and Diffusion-DPO trained with MG, MG+RS, and F-GPS using (a) Aesthetic Score and (b) ImageReward. Higher win rate indicates stronger preference under the corresponding quality evaluator.

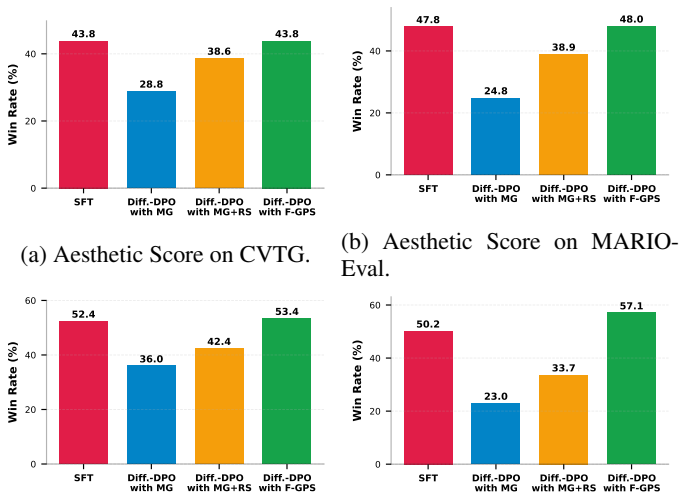


(a) Aesthetic Score.



(b) ImageReward.

Figure 25: **Quality evaluation for image composition on T2I-CompBench++.** Radar plots show category-wise results for SD v1.5, SDXL-Base-1.0, and SD3.5-Medium from left to right. We compare SFT and Diffusion-DPO trained with MG, MG+RS, and F-GPS across the ten T2I-CompBench++ categories. Overall, F-GPS maintains or improves quality across most categories and backbones, while MG often yields the weakest quality performance.



(a) Aesthetic Score on CVTG.

(b) Aesthetic Score on MARIO-Eval.

(c) ImageReward on CVTG.

(d) ImageReward on MARIO-Eval.

Figure 26: **Quality evaluation for visual text rendering on CVTG and MARIO-Eval.** We compare SFT and Diffusion-DPO trained with MG, MG+RS, and F-GPS using Aesthetic Score and ImageReward as quality evaluators. F-GPS matches or exceeds SFT across the four evaluations and consistently outperforms MG and MG+RS, indicating that the alignment gains do not come at the expense of overall generation quality.

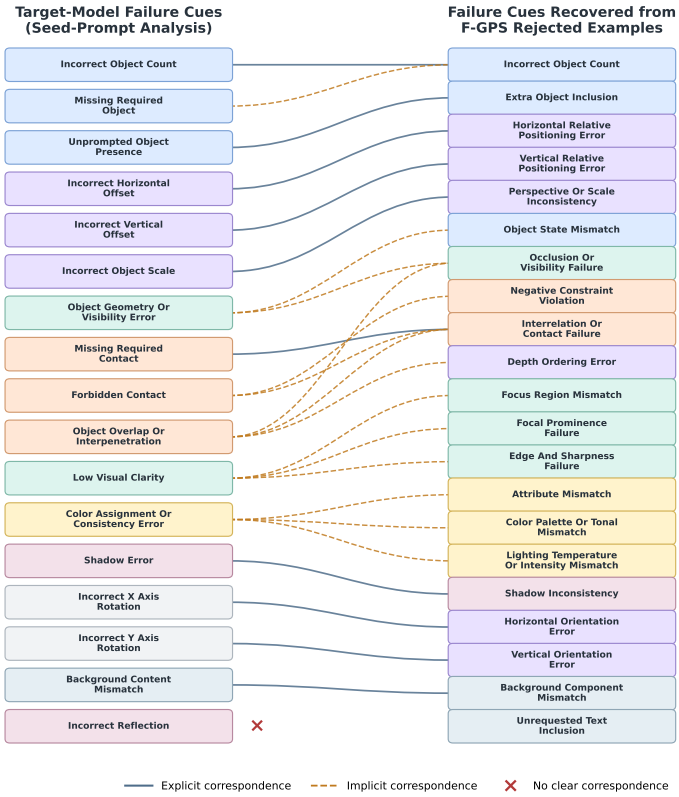


Figure 27: **Correspondence between target-model failure cues and those recovered from F-GPS rejected examples.** F-GPS recovers 16 of 17 atomic visual cues, corresponding to 94.1% failure-cue coverage.

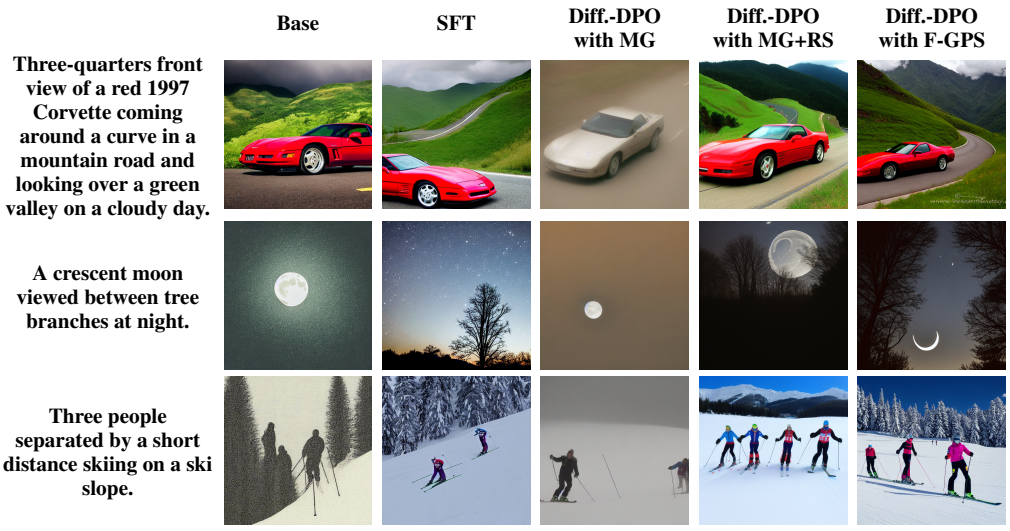


Figure 28: **Qualitative results on the TIFA benchmark for SD v1.5.** Each row highlights a different compositional requirement. **Top:** the prompt requires a red Corvette on a curved mountain road overlooking a green valley; F-GPS better preserves the specified vehicle appearance together with the surrounding road and landscape. **Middle:** the prompt requires a crescent moon viewed between tree branches at night; F-GPS captures both the crescent shape and the requested placement, whereas several alternatives render a circular or full moon or fail to satisfy the spatial relation. **Bottom:** the prompt specifies three people skiing with short separation between them; F-GPS better preserves the requested count and relative spacing.

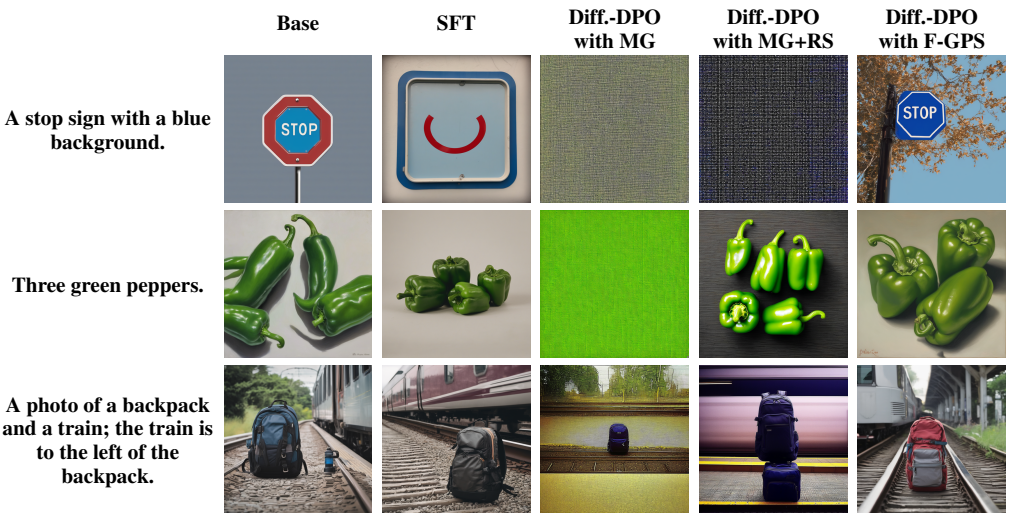


Figure 29: **Qualitative results on the TIFA benchmark for SDXL-Base.** Each row highlights a different compositional requirement. **Top:** the prompt requires a stop sign together with a blue background; F-GPS better preserves both the requested object identity and the specified background attribute, whereas the alternative outputs either miss the stop-sign structure or fail to reflect the intended background condition. **Middle:** the prompt specifies exactly three green peppers; F-GPS more accurately preserves both the required count and the object attribute, while alternative outputs exhibit incorrect counts, degenerate generations, or less faithful object configurations. **Bottom:** the prompt requires both a backpack and a train with the train positioned to the left of the backpack; F-GPS more closely reflects the requested entities and their relative arrangement, while the alternative outputs either weaken the spatial relation or fail to clearly satisfy the intended composition.



Figure 30: **Qualitative results for visual text rendering using SDXL-Base-1.0 across different training strategies.** Each row highlights a different text-rendering requirement. **Top:** the prompt requires the word "Travel" to appear inside a baggage trolley; F-GPS better preserves both the requested text content and its placement, while alternative outputs show weaker readability or incorrect text realization. **Middle:** the prompt requires a wall banner containing the phrase "Best Deal"; F-GPS better captures the intended wording and scene placement, whereas alternative generations produce less legible or less faithful text. **Bottom:** the prompt requires a popcorn machine labeled "Hot And Fresh"; F-GPS more reliably renders the requested phrase with better readability and object–text correspondence, while the other outputs exhibit text distortion or reduced accuracy.

J INSTRUCTIONS GIVEN TO THE FAILURE ANALYST

This section presents the prompts used to guide the failure analyst in distilling structured failures observed in the target T2I model into reusable failure programs. The prompts follow the successive stages of F-GPS: (1) identifying sample-specific coarse failure instances, (2) atomizing them into fine-grained visual factors, (3) filtering and consolidating these factors into reusable atomic visual cues, and (4) operationalizing the retained cues as parameterized failure programs.

Coarse Failure Mode Identification (Image Composition)

Task Overview
I will provide you with an image generated by an image-generation model along with the input prompt that produced it.

Your task is to:
(1) *Analyze the image thoroughly and describe what you observe.*
(2) *Identify all model failure modes specifically related to composition. Remember that the analysis should be done with respect to the composition task; also remember that composition includes everything, such as the number of objects, colors, text, and everything else.*
(3) *Present each failure mode strictly in the following format:* **Failure Mode Name: Description.**

Output nothing else besides the failure mode names and their descriptions. The description is not the philosophy around the failure; it should be a clear one-line statement with some instances from the image.

For example: Failures can include fewer or more objects than mentioned in the prompt, or failures with respect to the relative position of the objects, etc.

Be extremely meticulous—do not overlook even subtle or minute issues. Evaluate the image as an expert image analyst.

The prompt corresponding to the input image is: `[prompt]`

Figure 31: Instruction for coarse failure mode identification in the Image Composition.

Coarse Failure Mode Identification (Visual Text Rendering)

Task Overview
I will provide you with an image generated by an image-generation model along with the input prompt that produced it.

Your task is to:
(1) *Analyze the image thoroughly and describe what you observe.*
(2) *Identify all model failure modes specifically related to text rendering.*
(3) *Present each failure mode strictly in the following format:*

Failure Mode Name: Description.

Output nothing other than the failure mode names and their descriptions. The description should not discuss the philosophy behind the failure; it must be a clear, one-line statement with specific instances from the image.

For example, failures may include alignment issues, glyph merging, character- or word-level distortions, noise artifacts, and more.

Be extremely meticulous—do not overlook even subtle or minor issues. Evaluate the text as an expert image analyst.

The prompt corresponding to the input image is: `[prompt]`

Figure 32: Instruction for coarse failure mode identification in Visual Text Rendering.

Failure Mode Atomization (Image Composition)

Task Overview *I will provide you with:*

1) *An image generated by an image-generation model,*

2) *The input prompt used to generate it, and*

3) *A description of the failure modes identified in the image.*

Your task is to perform the following:
(1) *Analyze the image, the prompt, and the described failure modes in depth.*
(2) *Decompose each coarse failure mode into **fine-grained, low-level, programmable factors**—these should be atomic, objective, and measurable visual failure components.*
(3) *Be exhaustive: identify as many fine-grained factors as possible, including multi-level factors when appropriate.*
(4) *Only output factors that are **directly programmable or computable** (e.g., counting of objects, orientation, and relative position of objects, etc.).*
(5) *Maintain strict focus on technical, quantifiable image defects—avoid subjective or aesthetic assessments.*
(6) *Do not add new failures by yourself. Use the content shared with you as reference, and your task is only to atomize those coarse failures with proper descriptions.*
(7) *Write the definition of each failure mode in a generic manner, with no image-specific information. Do not refer to any object, background, or other specific content from the provided image or prompt in the definition of the failure mode. For example, while defining misalignment issues, you can write: "Objects appear to be shifted in the x-direction." Do not reference any specific object.*

Expert Analysis Required: *Think like an expert visual-forensics and image-quality analyst. Capture even subtle, or edge-case failures with precision.*

For your help, I am also sharing the details of the tools I have. I am using the MuLAN dataset, which contains the component layers of the scenes. Therefore, I can make changes in different layers to alter the overall composition of the scene. While thinking about programmable and quantifiable factors, you can refer to this detail of the tool.

The Prompt and Description are: `[prompt]`

Figure 33: Instruction for failure mode atomization in the Image Composition task.

⌨ Failure Mode Atomization (Visual Text Rendering)

Task Overview
I will provide you with:

1) *An image generated by an image-generation model,*
2) *The input prompt used to generate it, and*
3) *A description of the failure modes identified in the image.*

Your task is to perform the following:
(1) *Analyze the image, the prompt, and the described failure modes in depth.*
(2) *Decompose each coarse failure mode into **fine-grained, low-level, programmable factors**—these should be atomic, objective, and measurable.*
(3) *Be exhaustive: identify as many fine-grained factors as possible, including multi-level factors when appropriate.*
(4) *Only output factors that are **directly programmable or computable** (e.g., X-axis misalignment, Y-axis baseline drift, kerning variance, stroke thickness inconsistency, rotation deviation, aspect-ratio distortion, etc.).*
(5) *Maintain strict focus on technical, quantifiable image defects—avoid subjective or aesthetic assessments.*

Expert Analysis Required: *Think like an expert visual-forensics and image-quality analyst. Capture even subtle, minute, or edge-case failures with precision.*

The Prompt and Description are: `[prompt]`

Figure 34: Instruction for failure mode atomization in the Visual Text Rendering.

⌨ Failure Mode Filtering and Consolidation

Task Overview
You will act as a verification and filtering system for text-to-image generation failure modes.

Input Provided:
I will provide you with a list of failure modes and their descriptions, derived from analyzing 100 generated images. The list may include redundant entries, overlapping categories, vague items, or non-programmable failures.

Your Role:
Think like an expert in visual forensics, image-quality assessment, and multimodal model evaluation. Your job is to:

- *Identify and keep only the most relevant, distinct, and programmable failure modes.*
- *Remove redundancy, overlapping categories, and eliminate non-actionable failures.*
- *Ensure each retained failure mode is clearly defined, measurable, and useful for model benchmarking.*

Task Requirements:
Filter, refine, and output the best curated list. Output the list of failure modes in the following format:
`(1) Failure: Description`
Do not output anything else.

Data:
Failure Modes and Descriptions: `[prompt]`

Figure 35: Instruction for filtering and consolidation of relevant failure modes.

Visual Program Construction (Image Composition)

Task Overview
You are given a collection of failure modes identified from analyzing images generated by a model. Your task is to convert each failure description into a visual transformation program that can be applied to a fresh layered image to approximate that failure mode.

Output only the exact Python code for the visual program. Do not output explanations, markdown, or any additional text outside the code.

Rendering Context
I am using the MuLan-style layered image dataset with a custom rendering tool. Each image is decomposed into multiple RGBA layers. Layer 0 is the background, and all subsequent layers are foreground/object instance layers.
Each foreground layer contains a cropped object image, its alpha mask, bounding box, and editable transformation properties. These layers are rendered together to form the final composed image.
You should use the available layer properties wherever appropriate to reproduce the given failure modes. These include visibility, position, scale, rotation, flipping, color mode, saturation, brightness, object image content, alpha masks, and object bounding boxes.
However, do not randomly tweak properties without reasoning. For each failure mode, first determine whether the failure can realistically be generated using the available layer structure and rendering parameters. If the existing parameters are insufficient, you are allowed to introduce additional image-processing operations or helper logic using Python libraries such as PIL, NumPy, OpenCV, random, copy, or other suitable libraries.
The goal is to create the best possible visual transformation programs that approximate the intended failure modes in a natural and visually plausible way. It is not mandatory to only modify existing layer parameters; you may also edit RGBA image content directly, distort masks, crop objects, remove object parts, duplicate layers, disable layers, move objects, modify colors, manipulate lighting, add noise/blur, or alter the background.

Function Form
Each visual program must be written as a method compatible with the LayerEditor class:

```
def failure_mode_name(self, index: int,
transformation_parameter1=default_value,
  transformation_parameter2=default_value):
  # code
```

Requirements

1. *Use the function name corresponding to the provided failure mode name.*
2. *Include reasonable default values for transformation parameters.*

3. *The function should operate within the provided dataset/editor/rendering structure.*
4. *The parameter `index` refers to the target layer index whenever applicable.*
5. *Do not apply object-specific transformations to the background layer unless the failure mode specifically targets the background.*
6. *If the failure mode requires selecting other layers or objects, introduce randomness while ensuring the transformation remains valid.*
7. *Add randomness wherever useful, such as choosing target layers, selecting affected regions, choosing intensity values, deciding movement direction, selecting duplicated or removed objects, applying distortion strength, or affecting one or multiple layers.*
8. *Do not apply transformations uniformly or mechanically to every layer unless the failure mode specifically requires it.*
9. *The transformation should be visually meaningful and should approximate the failure mode as closely as possible.*
10. *If a failure mode cannot be represented exactly using the dataset structure or image-level transformations, provide the closest reasonable approximation using available tools.*
11. *The generated functions must be compatible with the LayerDataset, LayerRenderer, and LayerEditor setup described below.*
12. *I have a list of RGBA images named `images`; if you need new unprompted random objects for any visual program, randomly choose from that list.*

Important Rendering Note
The renderer currently renders one visible object per layer using `layer['instances'][0]`. Therefore, if you need to create additional visible object instances, prefer creating duplicate layers instead of only appending new dictionaries inside `layer['instances']`.

Correct approach for visible object duplication: create a new layer with its own `instances[0]`. Avoid relying only on `layer['instances'].append(...)`, because extra instances may not be rendered unless the renderer is modified.

Actual Dataset Structure
Each image has a structure similar to the following. Layer 0 is the background:

```
layer = {
  'index': 0,
  'file': Path('/path/to/image-layer_0.png'),
  'type': 'background',
  'enabled': True,
  'count': 1,
  'instances': [{
    'position': (0, 0),
    'scale': 1.0,
    'rotation': 0.0,
    'flip_horizontal': False,
    'flip_vertical': False,
    ...
  }]
}
```

Foreground/object instance layers have the following structure:

```
layer = {
  'index': i,
  'file': Path('/path/to/image-layer_i.png'),
  'type': 'instance',
  'enabled': True,
  'count': 1,
  'instances': [{
    'position': (0, 0),
    'scale': 1.0,
    'rotation': 0.0,
    'bbox': (x1, y1, x2, y2),
    'object_mask': numpy_boolean_array,
    'original_image': PIL.Image.Image,
    'image': PIL.Image.Image,
    ...
  }]
}
```

Field Meaning
The fields include layer index, file path, layer type, enabled flag, count metadata, position, scale, rotation, flip flags, color mode, color intensity, saturation boost, brightness adjustment, bounding box, object mask, original RGBA object image, and current editable RGBA object image. Image-level corruptions should usually modify `inst['image']`.

Coding Guidelines

1. *Always validate the layer index before applying changes.*
2. *Skip background layers unless the transformation is background-specific.*
3. *When duplicating an object, assign a new* `index` *and append it to* `self.dataset.layers`.
4. *When removing layers, reindex all layers afterward.*
5. *When editing object pixels, operate on* `inst['image']`.
6. *Preserve alpha channel when modifying RGBA images.*
7. *Ensure generated code is robust when* `bbox`, `image`, *or* `object_mask` *is None.*
8. *Keep the transformation randomized but controlled by function parameters.*
9. *For failures related to object contact, search layers to determine whether they are in contact before applying the visual program.*
10. `background_images` *contains multiple background images. Use it if needed.*

Tool Documentation
Below is the full tool documentation containing the classes for dataset structure and rendering. Your generated visual programs must align with this setup: [Full tool documentation content]

Data
Failure Modes and Descriptions: `[prompt]`

Figure 36: Instructions for converting curated failure mode descriptions into executable visual transformation programs for Image Composition Task.

Visual Program Critique

Task Overview
You previously generated this visual transformation program: `[code]`

The failure mode to approximate is: `[prompt]`

Your task is to critique this code. Specifically check:
(1) *Does it actually approximate the described failure mode visually?*
(2) *Are layer indices validated before use?*
(3) *Is randomness used appropriately, and not too uniformly?*
(4) *Does it handle* `None` *values for* `bbox`, `image`, *and* `object_mask`*?*
(5) *Are there any bugs or incorrect assumptions about the layer structure?*
(6) *Could the transformation be more realistic or visually accurate?*

Important Note:
Remember one thing: you have to stick to the definition of the failure mode. There can be multiple cases, but since these failure modes are fine-grained, do not force the model to generate a visual program that contains all cases, because those cases will be separate visual programs. Provide the critique while considering this point.

Output Requirement
Output only a structured critique, no code.

Figure 37: Instruction for critiquing generated visual transformation programs.

Visual Program Refinement

Task Overview
Your previous attempt was:

`[code]`

A critique of the previous attempt is:

`[feedback]`

Your task is to:
(1) *Generate an improved version addressing all critique points.*
(2) *Output only the exact Python code, with no explanations.*

Important Note:
Remember one thing: you have to stick to the definition of the failure mode. There can be multiple cases, but since these failure modes are fine-grained, do not force the model to generate a visual program that contains all cases, because those cases will be separate visual programs. Try to apply the critique considering this point.

Figure 38: Instruction for refining generated transformation programs based on critique feedback.

Visual Program Construction (Visual Text Rendering)

Task Overview
You are given a collection of failure modes identified from analyzing images generated by a model. Convert each failure description into a visual transformation program that can be applied to a fresh image to approximate that failure mode.
Output only the exact Python code for the visual program, nothing else.

Rendering Context:
I am using the Crello dataset with the CR renderer. It includes various properties as mentioned below. Try to tweak these parameters to replicate various types of failure modes.

Function Form:
Each program should be a function of the form:

```
def transform(example, transformation_parameters):
  # code
```

Use the function name same as the provided failure mode. Also assign some values to transformation_parameters on your own.
Here, `example` is the object having all the properties, and `transformation_parameters` are the knobs to control intensity or other properties of the transformation (include Crello/CR properties if needed).

Randomness Requirements:
Try to make the transformation as random as possible. For example, you can add randomness while choosing the target index or while choosing the number of indices, etc. Instead of applying the distortion to each index uniformly, add randomness in choosing the indices and number of operations.

Dataset Structure:
The structure of the dataset is shown below: `Dataset Structure`

Important: *Understand the task properly and the structure of the example datasets before creating the program.*

Example (for reference only):

```
import copy
import random

def char_level_drop_distortion(example, max_drops:  int = 3):
  '''Drops random characters from the text.'''
  ..............................
```

Tool Documentation:
[Full tool documentation content]

Data:
Failure Modes and Descriptions: `[prompt]`

Figure 39: Instructions for converting curated failure mode descriptions into executable visual transformation programs for Visual Text Rendering Task.

K PROMPT USED FOR OCR-BASED EVALUATION

We use the following prompt to evaluate OCR performance on generated images. The prompt instructs the model to strictly extract the visible text without introducing corrections or additional interpretations, ensuring a fair and consistent evaluation.

OCR Evaluation Prompt

OCR Task Instructions

Return the exact text visible in the image.

Requirements:

- *Output only the text present in the image.*
- *Do not correct spelling, grammar, or formatting.*
- *Do not add any commentary, explanations, or captions.*

Important Note:
The response is used directly for evaluation purposes. Return the exact text visible in the image without any correction to ensure a fair comparison.

Figure 40: Prompt used for OCR-based evaluation. The instructions enforce strict extraction of visible text without corrections or additional interpretation.

L PROMPTS USED FOR FAILURE ANALYSIS

Seed prompts for Image Composition

- `A pencil holder with more pens than pencils.`
- `A red bicycle against a blue wall.`
- `A little girl tuning a guitar before a concert in a dimly lit venue.`
- `A couple, with the taller one hugging the shorter one from behind.`
- `An animal with legs notably longer than a nearby person's.`
- `Three flowers on the ground:  one red, another yellow, and the third blue.`
- `There are two red flowers on the table and a few yellow flowers under the table.`
- `A painting hangs on the wall, a vase rests on the table, and sunlight streams through the window.`
- `A window consists of two panes of glass, the left one is broken but the right one is intact.`

- Two cats playing with a single ball.
- Four flowers are in full bloom, with two of them by the bench.
- Two orange pumpkins and four flying black bats.
- One sun setting behind two tall buildings.
- Three books are stacked on top of each other, with the red one at the bottom, the yellow one in the middle, and the blue one on top.
- Two red balloons and three white clouds floating in the blue sky.
- A tailless, not black, cat is sitting.
- A smiling girl with short hair and no glasses.
- A pair of glasses without lenses.
- A colorful skirt has an uncolorful hem.
- A plate with no food, only a fork and a knife.
- Four elephants, no giraffes.
- A vase with water, but no flowers to nourish.
- A tree with no leaves, standing in a field of green.
- A gigantic dog that is taller than the tree next to it.
- A table laden with apples and bananas, where all the fruits are green.
- In a square, several children are playing, each wearing a red T-shirt.
- A young man with a green bat and a blue ball.
- A dog chasing a cat.
- One cat is sleeping on the table and the other is playing under the table.
- A robot dancing in a futuristic cityscape.
- A balloon drifts gently up into the blue sky, with excited children watching from the ground.
- In the garden, there is a pack of dogs playing, the biggest one is red and the others are white.
- In the pond, two ducks swim near each other: the larger one has a bright green head, while the smaller one is all brown.
- Between the two cups on the desk, the taller one holds more coffee than the shorter one, which is half-empty.
- A silver spoon lies to the left of a golden fork on a wooden table.
- A big green apple next to a small red apple on a kitchen counter.
- A fluffy white cloud above a darker storm cloud at sunset.
- At a birthday party, all the balloons are red.
- In a field of flowers, every bloom is yellow, save for one that is blue.
- A birdcage with no door, yet no bird inside.
- A painting with no colors, only shades of gray.
- A garden full of flowers, yet not a single rose can be found.
- Two trees standing tall; one is leafy, and the other is not.
- A bustling city street with cars, but no bicycles.
- Five pencils on the desk, but no erasers.

- Two birds in the sky; the one on the left is flying, but the one on the right is not.
- Four cupcakes with sprinkles on a plate with two forks.
- Three kites soaring in the sky above two hills.
- Two apples on a kitchen counter.
- Three birds sitting on a wire.
- Four pencils in a cup.
- Five stars in the night sky.
- One cat napping in the sun.
- Two shoes by the front door.
- A bouquet of tulips sprouting from an open book.
- An octopus playing multiple instruments in an underwater band.
- A flamingo standing on one leg at a soda shop counter.
- A child with a leafy garland sitting on a tree branch.
- A monk with a wooden bracelet beside a tranquil pond.
- A lion with a golden chain roaring at sunset.
- An owl with a tiny book in a moonlit library.
- A green chalkboard with a detailed chalk drawing of a cityscape.
- A rotary dial telephone on a modern minimalist desk.
- A biplane parked in a field of wildflowers.
- Two old men with green hats sitting at a table.
- A girl with earrings, with her hands in her pockets, stands between two girls with handbags in their hands.
- A sitting puppy wearing star shaped glasses.
- Several blue birds perched on a branch, each with a red feather in their tails.
- A group of four children, two wearing hats and two holding balloons, stands near a fountain.
- Two cats with striped tails sitting side by side, one with a bow tie and the other without.
- Two parrots on a perch, one speaking into a miniature microphone.
- Seven pencils in a cup, each of a different color.
- A silk scarf draped over a marble statue in the foyer.
- A rubber-soled shoe stepping onto the smooth surface of a skateboard.
- A woolen sweater drying on a wooden rack in the sun, next to a linen shirt and denim jeans.
- A pair of sunglasses with shiny, metallic frames sitting on a glass table by the pool.
- A spherical balloon floating above a square table.
- A cylindrical vase sitting on a rectangular shelf.
- A circular clock hanging on a curved wall.
- An oval mirror leaning against a flat door.
- A spider is on the right of a crystal ball.
- There's a spider in the hovering crystal ball.
- A man stands in front of a small dog and blocks a flying football.

- A man stands on the left of a timid dog and blocks a flying football.
- Three curious monkeys.
- Five frustrated horses.
- Two excited elephants to the right of a lost giraffe.
- Five relieved turtles.
- Two excited pandas.
- Four bored boys.
- Four embarrassed zebras.
- Three jealous teachers.
- In the living room, the TV remote is not on the table.
- A rainy day with no umbrella by the door.
- A party with no candles on the cake.
- A bed without the usual cat sleeping on it.
- An office, the computer on, but no one there.
- There is no cup on the right side of a ceramic plate.
- A wide-brimmed hat not shading a napping cat casts a cool shadow.
- On a shelf hang a sweater and a windbreaker; the windbreaker is not on top of the sweater.
- There is an apple and two bananas on the table, neither of which is bigger than the apple.

Figure 41: Prompts used for failure analysis for Image composition task.

Seed prompts for Visual Text Rendering

- Black and white sign with the words "activeware" on a white background, wireframe, generative art.
- The slogan "aftnerne" is written on the schoolbag.
- Black and white sign with the words "alvoni" on a white background, wireframe, generative art.
- This car has a sign that reads "amprofon".
- "apazine" is written on the battery.
- Little bee holding a sign that says "bailup".
- In class, the teacher wrote the phrase "brahmaic" on the blackboard.
- At the airport, a sign that says "burwardeslyn".
- In a hospital, a sign that says "capitularis".
- A dark forest with only one light in the distance and the text "chapultepee".
- In class, the teacher wrote the phrase "chelonitis" on the blackboard.
- This cosmetic bottle says "churchilll".
- A lizard sitting on home plate at a baseball field with the words "copden" in a speech bubble.
- A cartoon of a cat with a thought bubble that says "cubelo".
- There is a "danakadan" sign in the hotel.
- There is a notice board next to the train station that reads "elitei".

- Small snail holding a sign that says "ensco".
- At the train station, a sign that says "galysheva".
- In the exhibition hall, a sign that reads "gamengame".
- Minimal sculpture of the word "garoff", made of light metallic iridescent chrome thin lines, 3D rendering, isometric perspective, super detailed, dark background.
- In the scenic spot, a sign that reads "hallatar".
- There is a sign saying "hammerclaw" in the mall.
- The slogan "harnetty" is posted on the bus stop.
- "heeresbekleidungsamt" slogan printed on school bus.
- A lizard sitting on home plate at a baseball field with the words "heyyyy" in a speech bubble.
- In a supermarket, a sign that says "josanas".
- Close up of a toothpaste tube figurine, 3D rendering, candy pastels, with the text "kangun" on the tube.
- A shot of a vine with the text "kokachin" sprouting, centered.
- The words "kornephoros" are written on the paper towel.
- Little frog holding a sign that says "linneas".
- Pillow in the shape of "malaefone", Alphabetism, fun jumbled letters, [close-up] bread, graphic art.
- "managat" reminder posted on the bus.
- Photos with "mediashopping" sign.
- A pumpkin with a beard, a monocle and a top hat with the text "minioudaki" in a speech bubble.
- Hubble and the Milky Way with the text "monaut".
- Billboard with "muthialpettah".
- A parrot on a pirate ship, a parrot wearing a pirate hat with the text "odnoin".
- A plane flies over the city with the words "ormelle" written in smoke trails.
- The slogan of "paikary" is written in the lottery station.
- "quercusglobulus" reminder posted on the bus.
- A newspaper headline read "qurniyya" and a photo showed a half-eaten pumpkin.
- "rimydis" reminder posted in the restaurant.
- There is a notice in the supermarket that says "saukrieg".
- Professionally designed logo for a bakery called "scilliano".
- A globe with the words "secciones" written in bold letters.
- At the train station, a sign that says "serranito".
- Professionally designed logo for a bakery called "siyaya".
- In a hospital, a sign that says "slevyas".
- Piggy holding a sign that says "spissoy".
- Grow in a pretty pot with a "sukhaybarat" sign.
- Scrabble board showing the words "taktai".
- A hastily handwritten note saying "thangundi" posted on the fridge.
- Photo of a helicopter with "thellaru" written on the side, landing in a valley.
- A shot of a vine with the text "thornborg" sprouting, centered.

- Billboard with "unveiledness".
- "vayalil" sign with home decor.
- A street sign on the street reads "wowomen".
- A t-shirt that says "zwangsanstalt".
- There is a sign saying "accept" in the mall.
- Close up of a toothpaste tube figurine, 3D rendering, with the text "assembly".
- The city of Toronto seen from the plane with the text "away".
- A dark forest with only one light and the text "boy".
- Little bee holding a sign that says "cell".
- Professionally designed logo for a bakery called "club".
- A photo of an aquarium with the words "construct".
- A lizard sitting on home plate with the words "debut" in a speech bubble.
- There is a "department" sign in the hotel.
- Black and white sign with the words "direct", wireframe.
- Professionally designed logo for a bakery called "effort".
- Billboard with "energy".
- A picture of the Earth with the words "facility".
- A t-shirt that says "feature".
- There is a notice in the supermarket that says "financial".
- In a supermarket, a sign that says "fly".
- There is a notice in the supermarket that says "help".
- A street sign reads "hospital".
- Scrabble board showing the words "individual".
- At the train station, a sign that says "lake".
- The words "leave" are written on the paper towel.
- A yellow saxophone in rainbow smoke with the words "london".
- Black and white sign with the words "look", wireframe.
- A detailed drawing with the text "lord".
- Little frog holding a sign that says "measure".
- Piggy holding a sign that says "notable".
- "october" generative art, sticky smoke made of dots.
- Small snail holding a sign that says "operate".
- "party" reminder posted on the bus.
- In class, the teacher wrote the phrase "popular" on the blackboard.
- At the train station, a sign that says "professional".
- The slogan "program" is posted on the bus stop.
- Photo of a helicopter with "public" written on the side.
- The sign "reach" hangs beside the swimming pool.
- At the airport, a sign that says "regular".
- Pillow in the shape of "religious", Alphabetism, graphic art.
- In a hospital, a sign that says "survive".

Figure 42: Prompts used for failure analysis for Visual text rendering task.