WHAT TO SHOW AND HOW TO SHOW: STRUCTURED FAILURE-GUIDED SUPERVISION FOR T2I ALIGNMENT

Pulkit Bansal
TCS Research, India
bansal.pulkit1@tcs.com

Vivek Srivastava
TCS Research, India
srivastava.vivek2@tcs.com

Shirish Karande
TCS Research, India
shirish.karande@tcs.com

ABSTRACT

Feedback-driven alignment has emerged as a primary mechanism for improving the controllability and reliability of generative models. While existing approaches focus on improving behavioral alignment through preference optimization, relatively little attention has been paid to how supervision design shapes the underlying representations that give rise to aligned behavior. This challenge is particularly pronounced in diffusion-based text-to-image (T2I) models, where errors are spatially distributed and arise from entangled visual representations, making it difficult to provide targeted corrective feedback analogous to language-based supervision. In this work, we propose **F-GPS** (Failure-Guided Preference Supervision), a framework that leverages structured and recurring failure patterns of diffusion models to construct targeted supervision signals for alignment. F-GPS identifies systematic failure modes and atomizes them into composable visual programs that synthesize controlled failure instances while preserving semantic content. By exposing failure-relevant ambiguities during training, the proposed approach encourages representations that better separate human-relevant visual distinctions without requiring large-scale human annotation or uncontrolled model sampling. We evaluate F-GPS on visual text rendering, a setting where failures are systematic and symbol-sensitive, and demonstrate that failure-guided supervision leads to improved prompt adherence and stronger alignment in text-sensitive image generation. Our results suggest that explicitly modeling structured failures provides an effective pathway for improving representational alignment in diffusion models through supervision design.

1 INTRODUCTION

Feedback-driven learning has become a central mechanism for aligning modern generative models with human intent. Beyond improving observable behavior, feedback implicitly shapes the internal representations learned by models, determining whether model features organize semantic and perceptual concepts in ways that correspond to human judgments. In LLMs, this paradigm has been largely explored along two complementary directions: training and inference-time feedback. At training time, models are optimized using explicit human preferences, instruction tuning, and reinforcement learning from human feedback (RLHF) (Ouyang et al., 2022; Bai et al., 2022; Wang et al., 2023). At inference time, techniques such as iterative self-refinement, critique-and-revision, and external guidance allow models to incorporate corrective textual feedback during generation without modifying model parameters (Madaan et al., 2023; Shinn et al., 2023; Yao et al., 2022). These approaches are effective for language models because language operates in a discrete symbolic space, where errors can often be localized to specific tokens or reasoning steps. This property allows feedback to be expressed directly by specifying what should be changed in the output, and has contributed to improved controllability and reliability of aligned language models (Wei et al., 2022; Wang et al., 2023).

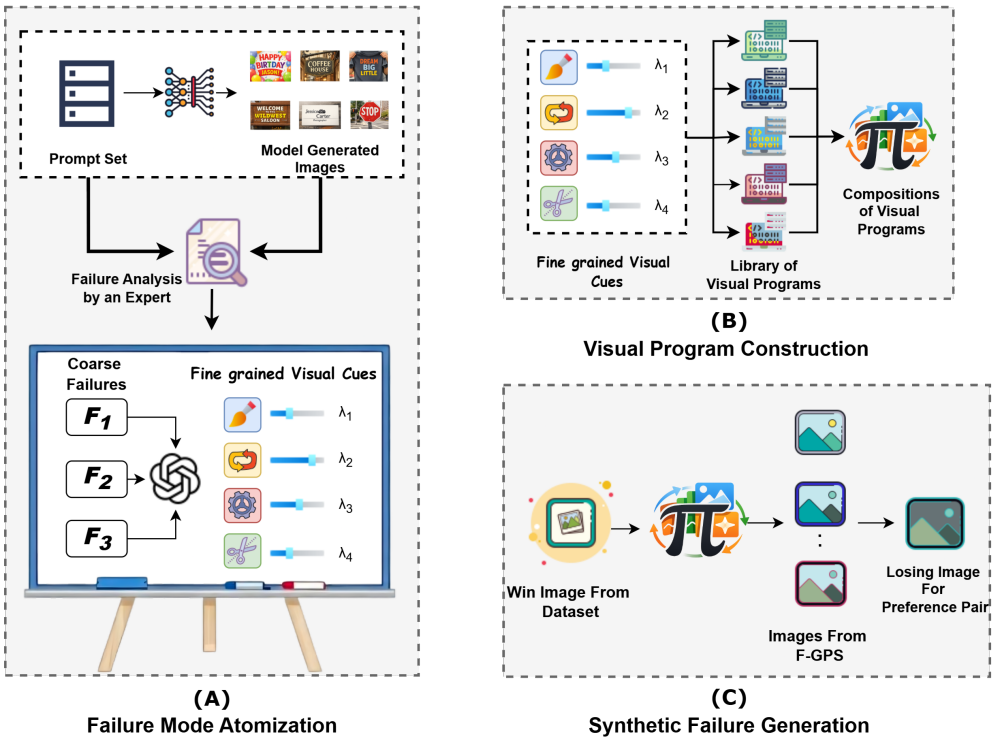


Figure 1: **Architecture of the F-GPS framework.** Model-generated images are first analyzed by an expert to identify coarse failure modes. These failures are then translated into visual programs to construct a reusable library. Synthetic failure generation composes visual programs from this library to produce the losing image in each preference pair.

Extending this paradigm to visual generation, however, presents substantially greater challenges. Diffusion-based text-to-image (T2I) models have achieved remarkable progress in generating high-quality images from natural language prompts (Saharia et al., 2022; Betker et al., 2023; Nichol et al., 2021; Rombach et al., 2022; Podell et al., 2023). Despite these advances, widely deployed models, particularly compact variants such as SDXL-Base, continue to exhibit prompt-image misalignment, where generated images fail to follow fine-grained instructions specified in the input prompt. Unlike language models, diffusion models operate in high-dimensional continuous pixel spaces, where visual errors are spatially distributed and entangled across color, texture, geometry, and object relationships, and images lack explicit compositional units, making it difficult to isolate individual visual deviations (Sahin et al., 2024; Zheng et al., 2024; Kamath et al., 2024; Zeng et al., 2024). As a result, corrective feedback must operate indirectly by reshaping high-dimensional latent representations, where misalignment often arises from entangled feature representations rather than isolated output errors. This property make it difficult to provide direct and interpretable corrective feedback analogous to textual supervision in LLMs as discussed earlier.

To address this challenge, recent work has explored preference-based alignment strategies for diffusion models, including Reinforcement Learning from Human Feedback (RLHF) Black et al. (2023); Liang et al. (2024); Fan et al. (2023b), Diffusion DPO Wallace et al. (2024), Diffusion KTO Li et al. (2024), and DSPO Zhu et al. (2025). These methods operationalize feedback using binary preference signals to indicate which output better satisfies human intent. To reduce reliance on external annotation, recent work has explored deriving training feedback directly from model-generated samples (Yuan et al., 2024; Alemohammad et al., 2024; Hu et al., 2023a; Chen et al., 2025).

At a broader level, feedback-driven learning raises a fundamental supervision design question: ***what*** *training signals should be presented to the model, and* ***how*** *these signals should be constructed to induce effective behavioral and representational change*. This question is central to training-time feedback methods, where the choice of supervision data, example construction, and feedback representation directly determines the quality of the learned policy (Bengio et al., 2009). Poorly

designed feedback can lead to inefficient learning, biased updates, or overfitting to spurious patterns, even when the underlying optimization objective is well defined.

In practice, most existing alignment approaches rely on either human-annotated preference datasets or automatic scoring models such as PickScore (Kirstain et al., 2023). However, both strategies suffer from important limitations. Human annotation is expensive, subjective, and often noisy, with prior work reporting substantial inconsistency in collected preferences (Kittur et al., 2008; Long et al., 2013; Chen et al., 2013; Zheng et al., 2023). Automatic scoring models, while more scalable, inherit biases from their training distributions and exhibit limited robustness across diverse prompts and visual concepts. As a result, the supervision signals used for alignment frequently remain weak or unreliable, limiting training efficiency and alignment performance (Yang et al., 2024c;b; Chowdhury et al., 2024).

On the other end, methods that rely on self-generated samples as feedback signals avoid external annotation but introduce a different limitation: feedback selection becomes coupled to the model's own sampling behavior and offers limited control over which error patterns are emphasized during training. In practice, negative samples are often selected in a generic manner, causing many training examples to either already satisfy the prompt or provide weak contrastive supervision.

At the same time, empirical analyses demonstrate that diffusion model failures are not random, but exhibit recurring and structured patterns across prompts and models, including systematic attribute binding errors, spatial inconsistencies (Sagar et al., 2024; Huang et al., 2023). These recurring failures suggest that misalignment often arises from systematic representational deficiencies, where attributes, spatial relations, or symbolic structures are not cleanly separated in the learned representation space. Despite this structure, existing alignment pipelines do not explicitly leverage failure patterns when constructing feedback signals.

Motivated by these observations, we propose **F-GPS** (**F**ailure-**G**uided **P**reference **S**upervision), a framework that transforms structured failures of diffusion models into targeted supervision signals for alignment (refer Figure 1). F-GPS explicitly identifies recurring failure modes from model outputs and atomizes them into programmable visual cues, which are formalized as a library of composable visual programs that can be applied to high-quality reference images. This design enables controlled instantiation of specific failure behaviors while preserving semantic content, exposing representational ambiguities during training and encouraging the model to organize latent representations in ways that better reflect human-relevant visual distinctions. To evaluate the effectiveness of failure-guided supervision in a setting where structured errors are especially pronounced, we study visual text rendering, a task in which failures are systematic and symbol-sensitive, and where small visual deviations can lead to severe semantic errors. This makes text rendering a stringent testbed for assessing whether explicitly modeling and synthesizing structured failures can yield informative and targeted supervision, without requiring exhaustive sampling from the model itself.

Our key contributions are summarized as follows:

- **Failure-guided preference supervision.** We introduce **F-GPS**, a framework that leverages structured diffusion model failure patterns as targeted feedback signals to enrich preference supervision and improve representational alignment between diffusion model features and human perceptual distinctions.
- **Compositional failure synthesis.** We construct a library of visual programs that enables controllable and semantically preserving synthesis of diverse failure instances through the composition of fine-grained visual transformations.
- **Failure-guided supervision for text rendering.** We validate our framework on visual text rendering, demonstrating that explicitly modeling and synthesizing text-specific failure patterns leads to stronger alignment in text-sensitive image generation.

2 Related Work

2.1 Preference Alignment in T2I Generation

Diffusion-based text-to-image models have demonstrated strong generative capabilities (Saharia et al., 2022; Betker et al., 2023; Ho et al., 2020; Nichol et al., 2021; Rombach et al., 2022; Podell

et al., 2023). However, aligning their outputs with human intent remains a central challenge, particularly for complex or subjective prompts (Wallace et al., 2024; Xu et al., 2023). Reinforcement Learning from Human Feedback (RLHF) Christiano et al. (2017); Stiennon et al. (2020) has been widely adopted for this purpose and recently extended to diffusion models (Black et al., 2023; Liang et al., 2024; Fan et al., 2023b;a). Despite its effectiveness, RLHF often incurs substantial computational overhead and is susceptible to reward modeling biases and instability.

To mitigate these limitations, reward-free and direct preference optimization approaches have been proposed, including Diffusion-DPO Wallace et al. (2023), D3PO Yang et al. (2024a), Diffusion-RPO Gu et al. (2024), Diffusion-KTO Li et al. (2024), InPO Lu et al. (2025), DenseReward Yang et al., and DSPO Zhu et al. (2025). SmPO-Diffusion Lu et al. further models preference distributions to improve robustness under preference variability. Other works focus on improving pair selection and supervision efficiency, such as quality-aware pair ranking Singh et al., personalized preference distillation (PPD) Dang et al. (2025), and CaPO Lee et al. (2025), which calibrates preferences across multiple reward models and performs frontier-based pair selection.

While these approaches improve optimization stability and scalability, they primarily assume the availability of high-quality preference datasets. As discussed in our introduction, the question of *what* examples to show and *how* to construct informative supervision remains largely orthogonal to the optimization objective itself. Our work complements these methods by focusing on structured failure-aware data construction for preference alignment.

2.2 Failure Mode Analysis in Generative Models

A growing body of work shows that generative model failures exhibit structured and recurring patterns rather than isolated artifacts. In diffusion-based T2I systems, systematic failure regions have been identified through adversarial prompt and latent space search, as demonstrated by SAGE, which uncovers diverse semantic and compositional errors that do not emerge under random sampling (Liu et al., 2024b). Diagnostic benchmarks such as T2I-CompBench Huang et al. (2023) and TIFA Hu et al. (2023b) further categorize common failure types including attribute binding errors, spatial inconsistencies, miscounting, and missing visual entities. Complementary to benchmark-driven analysis, LLM-assisted red-teaming approaches Sagar et al. (2024) actively explore diffusion failure landscapes, reinforcing the observation that model errors concentrate in structured regions of the generation space. Beyond failure discovery, FineGRAIN Hayes et al. (2025) introduces a large-scale structured evaluation framework that explicitly categorizes dozens of fine-grained failure modes, such as attribute fidelity, object counting, and compositional errors, revealing consistent and nuanced failure patterns across state-of-the-art T2I models.

Together, these studies establish that diffusion model failures are systematic, interpretable, and recurrent, providing a foundation for structured modeling of the failure space. Our work builds on this observation by operationalizing failure patterns as composable visual transformations, enabling explicit representation and controlled instantiation of recurring failure behaviors.

2.3 Training with Failure-Driven and Synthetic Supervision

Recent work has explored training paradigms that leverage feedback derived from model-generated outputs to reduce reliance on static human-annotated datasets. In language models, Learning from Mistakes (LeMa) An et al. (2023) augments training with mistake–correction pairs, while Self-Play Fine-Tuning (SPIN) Chen et al. (2024b) contrasts self-generated responses across iterations. Related approaches enable reward signal generation Yuan et al. (2024) or internalized correction mechanisms (Upadhyaya & Sridharamurthy, 2024). In diffusion models. SPIN-Diffusion Yuan et al. (2024) adapts self-play to image generation by contrasting samples from successive checkpoints. Group Preference Optimization (GPO) Chen et al. (2025) optimizes preference objectives over groups of sampled candidates, while SIMS Alemohammad et al. (2024) introduces negative guidance signals to steer diffusion trajectories. Self-guided diffusion models Hu et al. (2023a) explore self-supervised guidance strategies, and iterative prompt relabeling Chen et al. (2024a) refines textual supervision using generated images.

These approaches demonstrate the value of model-generated feedback for scalable training. Our work complements this direction by introducing a structured and interpretable mechanism for syn-

thesizing failure-driven supervision. Rather than treating generated samples as black-box negatives, we explicitly model recurring failure patterns using composable visual programs and select representative examples through coverage-aware coreset construction. This enables controlled failure instantiation and more targeted preference supervision.

3 METHODOLOGY

We present **F-GPS**, a failure-guided preference supervision framework for aligning diffusion-based text-to-image models. It constructs targeted preference supervision by identifying recurring model failure patterns and instantiating them through controlled visual programs. The overall pipeline consists of three stages: (1) Failure Mode Atomization, (2) Visual Program Construction (3) Synthetic Failure Generation.

3.1 NOTATION AND PROBLEM STATEMENT

Let $\mathcal{D}_{\text{ref}} = \{(c_i, x_i^+)\}_{i=1}^{N_{\text{ref}}}$ denote a reference set of high-quality (preferred) images x_i^+ paired with prompts c_i. Our goal is to construct a set of negative samples $\mathcal{S}^- = \{x_j^-\}_{j=1}^n$ that represent structured model failures, such that they (i) cover the failure space, (ii) are interpretable and parameterized, and (iii) form effective preference pairs (x^+, x^-) for alignment.

For a given prompt c, we define the failure set as

$$\mathcal{F}_c \triangleq \{x \mid x \text{ violates human preference for } c\},$$

and the global failure set as $\mathcal{F} = \bigcup_c \mathcal{F}_c$. Using F-GPS, we aim to construct a set of structured negative samples that approximates the failure set $\mathcal{F}$ through controlled and interpretable transformations.

3.2 FAILURE MODE ATOMIZATION

We begin by sampling a modest set of prompts $\mathcal{P}_c = \{c^{(t)}\}_{t=1}^T$ from a prompt space. For each prompt, we generate multiple model outputs, which are then presented to experts for inspection. The experts examine to identify *recurring and systematic failure behaviors*, rather than isolated artifacts. These behaviors are recorded as a set of coarse-grained failure modes, each corresponding to a qualitatively distinct type of model failure.

Following this coarse-level identification, experts further *atomize* the observed failures into a collection of fine-grained visual cues that characterize how failures manifest at a more detailed level. Across all identified failure modes, this process yields a finite set of fine-grained cues

$$\mathcal{A} \triangleq \{a_1, \ldots, a_L\}.$$

Each cue $a_\ell \in \mathcal{A}$ is associated with a finite parameter set Λ_ℓ together with a parametric description of the corresponding visual effect. This atomization captures the fact that a single output may exhibit multiple, overlapping distortions, and that a high-level failure mode can arise from several distinct low-level phenomena. The resulting cue-level descriptions provide an interpretable and fine-grained representation of how failures manifest in practice.

3.3 VISUAL PROGRAM CONSTRUCTION

We convert each fine-grained visual cue $a_\ell \in \mathcal{A}$ into a parameterized *visual program* that explicitly encodes the corresponding failure behavior as an executable image transformation. Formally, each visual program is defined as

$$T_{\ell,\lambda} : \mathcal{X} \to \mathcal{X}, \quad \lambda \in \Lambda_\ell,$$

where ℓ indexes the failure cue and λ denotes a set of interpretable control parameters governing the strength, spatial extent, and appearance of the transformation.

Each visual program introduces a specific failure cue through controlled modifications of a preferred image. The transformations are parameterized and stochastic, enabling diverse instantiations of the same failure pattern. They are designed to preserve the underlying semantic content, so that

Figure 2: Horizontal Misalignment Transformation that introduces random horizontal offsets to characters, simulating misaligned text by perturbing their x-axis positions.

```python
import copy
import random

def horizontal_misalignment(example, transformation_parameters=None):
    """
    Introduces horizontal misalignment by deviating characters along the x-axis from their original positions.
    """
    if transformation_parameters is None:
        transformation_parameters = {
            "max_x_offset": 10  # Maximum horizontal deviation in pixels
        }

    new_example = copy.deepcopy(example)
    max_x_offset = transformation_parameters["max_x_offset"]

    for i, text in enumerate(new_example["text"]):
        if text:  # Check if there's text in the element
            text_length = len(text)
            x_offsets = [
                random.uniform(-max_x_offset, max_x_offset) for _ in range(text_length)
            ]
            new_example["x_offset"][i] = x_offsets  # Update x_offset map with random deviations

    return new_example
```

Figure 3: Kerning and spacing irregularity transformation used to introduce uneven spacing between characters during perturbation.

```python
def kerning_and_spacing_irregularity(example, transformation_parameters=None):
    """
    Introduces uneven spacing between characters or words.
    """
    new_example = copy.deepcopy(example)
    transformation_parameters = transformation_parameters or {"max_offset": 5.0, "irregularity_probability": 0.5}

    for i in range(new_example["length"]):
        if new_example["type"][i] == 1:
            text_length = len(new_example["text"][i])

            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
            ]
            new_example["letter_spacing"][i] = sum(letter_spacing_values) / len(letter_spacing_values)

    return new_example
```

the resulting image differs from the original primarily along the targeted failure dimension. This construction produces interpretable preference pairs and supports systematic generation of diverse failure variations from a fixed set of program operators. Example visual program is given below in Figure 2 and Figure 3 with more examples in Table 1 and the Appendix.

3.4 Synthetic failure generation

Given a preferred image $x^+ \in \mathcal{X}$, we generate multiple negative samples by applying sampled compositions of visual programs. Let $\pi = ((\ell_1, \lambda_1), \ldots, (\ell_r, \lambda_r))$ denote a sequence of cue–parameter pairs sampled from the set of available programs, with composition length $r \geq 1$. The composed

transformation associated with π is defined as

$$T_\pi \triangleq T_{\ell_1,\lambda_1} \circ \cdots \circ T_{\ell_r,\lambda_r},$$
$$x^- = T_\pi(x^+),$$

which yields a negative sample that deviates from x^+ along a specific combination of fine-grained visual cues. For each preferred image x^+, we sample n compositions $\Pi(x^+)$ and obtain a corresponding set of n negative samples

$$\mathcal{N}(x^+) \triangleq \{\, T_\pi(x^+) \mid \pi \in \Pi(x^+),\ |\Pi(x^+)| = n \,\}.$$

This one-to-many construction reflects the fact that a single model output may admit multiple distinct failure realizations, and allows different failure factors to be instantiated and combined in a controlled manner. Given a reference set $\mathcal{D}_{\text{ref}}$ of prompt–image pairs (c, x^+), the full synthetic failure dataset is defined as

$$\mathcal{S} \triangleq \bigcup_{(c,x^+)\in\mathcal{D}_{\text{ref}}} \mathcal{N}(x^+).$$

By sampling compositions and varying their parameters, $\mathcal{S}$ provides representative coverage of the structured failure space identified through expert analysis. As a result, preference learning can contrast each preferred image against targeted negatives, encouraging suppression of recurring failure behaviors rather than reinforcing outputs the model already handles well.

4 EXPERIMENTS

We evaluate our method on the visual text rendering task. This task represents a structured evaluation regime that requires precise adherence to discrete symbolic content. Evaluating this task allows us to assess the robustness of our method across diverse alignment scenarios. Refer to the Appendix for additional experimental details and examples.

Dataset and Models. We construct a text-focused benchmark using the `Crello` dataset (Yamaguchi, 2021). We retain only samples whose absolute difference between image width and height is less than or equal to 50 pixels and remove samples without rendered text, resulting in 6,218 samples. The Crello dataset provides background images and textual content as separate annotations. To obtain natural instruction-style prompts for text-to-image generation, we synthesize prompts using the `Qwen3-VL-4B-Instruct` Team (2025) model by providing the background image, target text, and layout metadata as inputs, and generating aligned textual descriptions.

To account for text rendering limitations of compact diffusion backbones, we restrict long textual content and select the two text strings with the largest font sizes per image. Text-overlay images are generated using the `CyberAgentAILab/cr-renderer` tool, which we extend to support character-level synthesis by deriving spatial and style attributes (x-offset, y-offset, font size, font style etc.) from the provided word-level annotations. Text rendering experiments are conducted exclusively on the SDXL-Base model. For training we have used 5000 samples from this dataset.

For performing the failure atomization task and constructing visual programs based on the identified failure modes, we utilize `GPT-4o` Achiam et al. (2023) as an expert. Specifically, `GPT-4o` is employed to systematically analyze model-generated outputs, decompose observed errors into fine-grained failure categories, and assist in formulating corresponding visual distortion programs. This enables a structured and scalable pipeline for transforming failures into actionable programs. Refer Appendix for the prompts used and instructions given to the expert for failure analysis.

Evaluation Protocol. Text rendering performance is evaluated using 500 prompts sampled from CVTG benchmark (Du et al., 2025) and MARIO-Eval (Chen et al., 2023) respectively. We report Exact Match (EM), Normalized Exact Match (NEM), Partial String Score (PSS), Word Precision (WP), Word Recall (WR), and Word F1 (WF1) scores. For OCR based evaluation, text recognition is performed using `llava-v1.5-13b` (Liu et al., 2024a).

Visual Programs. For text rendering supervision, we utilize a structured pool of visual programs obtained through F-GPS with `GPT-4o` acting as an expert failure analyst. These programs are derived from systematic failure atomization and target both character-level recognition errors and layout-level distortions. The resulting pool includes: (1) horizontal misalignment, (2) vertical misalignment, (3) angular distortion, (4) line thickness inconsistency, (5) aspect ratio distortion, (6)

Table 1: F-GPS generated negative samples for different distortion types.

Distortion Type	Original Image	Distorted Image
Horizontal Misalignment		
Vertical Misalignment		
Angular Distortion		
Multilingual Confusion		

kerning and spacing irregularity, (7) text content mismatch, (8) character omission, (9) unintended glyph addition, (10) font style inconsistency, (11) readability deviation in small text, (12) multilingual confusion, (13) word break errors, (14) random character sequencing, (15) inconsistent letter capitalization, (16) scaling distortion, and (17) resolution drop.

Implementation Details. For each reference image, we generate 50 synthetic negative samples using a pool of text-specific visual programs. From these, we randomly sample one candidate to serve as the negative output in the constructed preference pair. We fine-tune the model using a LoRA-based configuration with a batch size of 16.

Baselines. We compare F-GPS against several training strategies. The unaligned SDXL-Base model is included as the base reference. We first consider standard Supervised Fine-Tuning (SFT). For preference optimization, we evaluate Diffusion-DPO and DSPO under three configurations each: (i) the standard setup, where model-generated samples are used as negative examples; (ii) a shuffled variant, where the same negative model generated samples are randomly shuffled; and (iii) Diffusion-DPO/DSPO+F-GPS, where negatives are replaced with F-GPS-generated samples.

5 RESULTS

Table 2: Performance evaluation on the CVTG dataset across different training strategies.

Method	ckpt	Metrics					
		EM	NEM	PSS	WP	WR	WF1
Base	-	24.20	26.00	78.49	47.81	51.51	47.34
SFT	700	26.80	29.40	80.11	51.11	54.20	50.38
DPO	300	23.20	26.40	78.13	48.65	50.69	47.48
DPO+shuffled	300	28.80	31.20	83.08	55.96	57.44	53.95
DPO+Ours	900	30.00 (+6.80 ↑)	32.20 (+5.80 ↑)	84.28 (+6.15 ↑)	54.92 (+6.27 ↑)	63.14 (+12.45 ↑)	55.69 (+8.21 ↑)
DSPO	500	26.40	29.00	82.69	53.71	56.86	52.71
DSPO+shuffled	300	27.00	28.80	80.61	52.32	56.09	51.83
DSPO+Ours	600	27.20 (+0.80 ↑)	29.80 (+0.80 ↑)	81.03 (-1.66 ↑)	54.12 (+0.41 ↑)	58.16 (+1.3 ↑)	53.35 (+1.52 ↑)

Table 3: Performance evaluation on the MARIO-Eval dataset across different training strategies.

Method	ckpt	Metrics					
		EM	NEM	PSS	WP	WR	WF1
Base	-	21.98	24.40	79.13	50.64	49.09	47.96
SFT	300	21.40	24.00	80.73	51.78	50.17	48.85
DPO	300	21.60	23.00	76.17	45.71	44.33	42.71
DPO+shuffled	400	22.60	25.00	80.22	52.37	48.71	48.53
DPO+Ours	400	27.20 (+5.6 ↑)	30.20 (+7.20 ↑)	81.54 (+5.37 ↑)	57.30 (+11.59 ↑)	54.48 (+10.15 ↑)	54.29 (+11.58 ↑)
DSPO	300	24.00	25.80	80.28	52.06	49.57	48.76
DSPO+shuffled	500	24.00	26.20	80.84	53.38	59.94	50.57
DSPO+Ours	400	27.40 (+3.40 ↑)	29.40 (+3.60 ↑)	80.85 (+0.57 ↑)	55.17 (+3.11 ↑)	52.34 (+2.77 ↑)	51.95 (+3.19 ↑)

5.1 QUANTITATIVE RESULTS

Tables 2 and 3 demonstrate consistent improvements across both datasets. Across CVTG and MARIO-Eval, Diffusion-DPO+Ours consistently yields the strongest gains, indicating improved alignment between generated outputs and textual prompts under preference optimization. DSPO+Ours also improves performance across all metrics, but with comparatively smaller gains, suggesting more conservative optimization behavior while still maintaining consistent benefits. Overall, the improvements are more pronounced in word-level metrics compared to exact or normalized matching metrics, indicating that the proposed method primarily enhances token-level alignment while also improving structural correctness. This suggests that the optimization primarily improves local textual fidelity within generated outputs rather than only global matching behavior. We additionally report loss profiling plots (see Fig. 4) to illustrate the impact of using F-GPS-generated samples as negatives. Compared to the near-zero skew observed when using model-generated negatives, incorporating F-GPS samples shifts the loss distribution closer to 0.5. This behavior supports our hypothesis that model-generated negatives contain limited informative signal for preference alignment, whereas F-GPS-derived negatives provide more meaningful supervision and stronger optimization gradients.

5.2 QUALITATIVE ANALYSIS

Table 4 presents qualitative comparisons across SDXL-Base, SFT, Diffusion-DPO with model generated negatives, Diffusion-DPO with shuffled model generated negatives, and Diffusion-DPO+F-GPS configurations. In the first row, only Diffusion-DPO+F-GPS successfully renders the target text clearly and completely. While Diffusion-DPO and SFT attempt to generate some characters, the rendered text suffers from poor contrast and noise interference, making it largely unreadable and visually inconsistent. In the second row, SDXL-Base produces heavily noisy outputs with distorted text. SFT manages to render several characters correctly but fails to generate the character

Table 4: Qualitative comparison. The prompts used are: (i) *"A baggage trolley has 'Travel' written on the inside"* (top-row) and (ii) *"A banner on the wall says 'Best Deal' (bottom-row)"*.

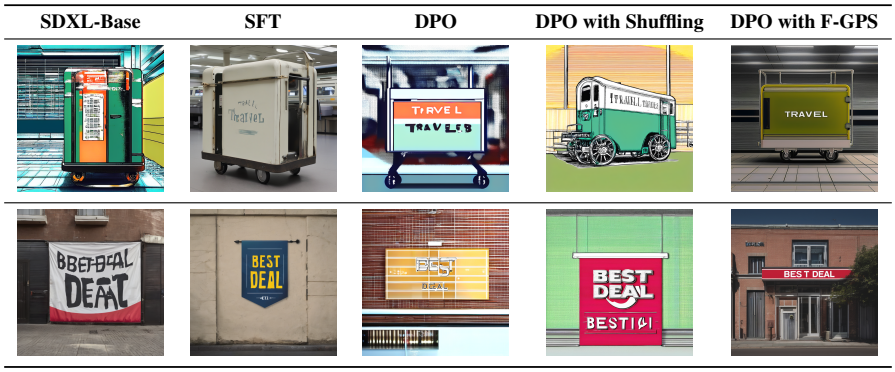
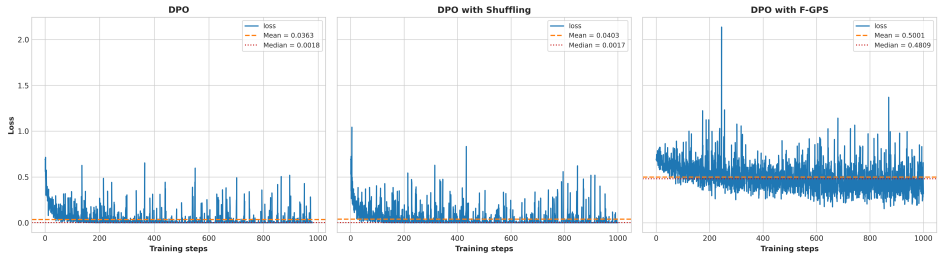
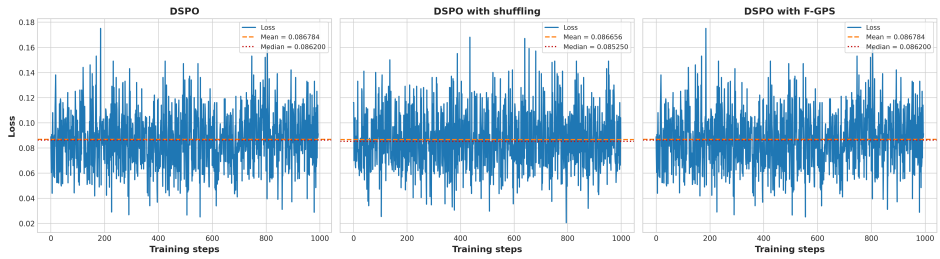


Figure 4: Training loss for Diffusion-DPO and DSPO with different configurations of the dataset.



(a) Diffusion-DPO



(b) DSPO

"A" accurately. Both Diffusion-DPO and Diffusion-DPO with shuffling struggle with noise-induced distortions, resulting in incomplete or malformed text. In contrast, Diffusion-DPO+F-GPS generates clean, well-structured text with all characters rendered accurately and with consistent shape, demonstrating improved character-level fidelity. Refer to Appendix for more qualitative results.

6 Conclusion

We presented F-GPS, a failure-guided preference supervision framework for improving alignment in diffusion-based text-to-image models. The method transforms structured model failures into targeted supervision signals using composable visual programs, enabling effective preference learning without relying on large-scale human annotations. Experiments on visual text rendering show consistent improvements across datasets and evaluation metrics. These results indicate that leveraging failure patterns provides more informative supervision than relying solely on model-generated negatives. The approach is simple to integrate into existing preference optimization pipelines. Overall, our findings suggest that explicit failure-driven supervision is a promising direction for improving text fidelity and structural alignment in generative models, and may extend to other fine-grained visual generation tasks.

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.

Sina Alemohammad, Ahmed Imtiaz Humayun, Shruti Agarwal, John Collomosse, and Richard Baraniuk. Self-improving diffusion models with synthetic data. *arXiv preprint arXiv:2408.16333*, 2024.

Shengnan An, Zexiong Ma, Zeqi Lin, Nanning Zheng, Jian-Guang Lou, and Weizhu Chen. Learning from mistakes makes llm better reasoner. *arXiv preprint arXiv:2310.20689*, 2023.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ouyang, Juntang Zhuang, Joyce Lee, Yufei Guo, et al. Improving image generation with better captions (2023). *URL https://cdn. openai. com/papers/dall-e-3. pdf*, 2023.

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

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.

Renjie Chen, Wenfeng Lin, Yichen Zhang, Jiangchuan Wei, Boyuan Liu, Chao Feng, Jiao Ran, and Mingyu Guo. Towards self-improvement of diffusion models via group preference optimization. *arXiv preprint arXiv:2505.11070*, 2025.

Xi Chen, Paul N Bennett, Kevyn Collins-Thompson, and Eric Horvitz. Pairwise ranking aggregation in a crowdsourced setting. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pp. 193–202, 2013.

Xinyan Chen, Jiaxin Ge, Tianjun Zhang, Jiaming Liu, and Shanghang Zhang. Learning from mistakes: Iterative prompt relabeling for text-to-image diffusion model training. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 2937–2952, 2024a.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint arXiv:2401.01335*, 2024b.

Sayak Ray Chowdhury, Anush Kini, and Nagarajan Natarajan. Provably robust dpo: Aligning language models with noisy feedback. *arXiv preprint arXiv:2403.00409*, 2024.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Meihua Dang, Anikait Singh, Linqi Zhou, Stefano Ermon, and Jiaming Song. Personalized preference fine-tuning of diffusion models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 8020–8030, 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.

Ying Fan, Olivia Watkins, Yuqing Du, Hao Liu, Moonkyung Ryu, Craig Boutilier, Pieter Abbeel, Mohammad Ghavamzadeh, Kangwook Lee, and Kimin Lee. Dpok: Reinforcement learning for fine-tuning text-to-image diffusion models. *Advances in Neural Information Processing Systems*, 36:79858–79885, 2023a.

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, 2023b.

Yi Gu, Zhendong Wang, Yueqin Yin, Yujia Xie, and Mingyuan Zhou. Diffusion-rpo: Aligning diffusion models through relative preference optimization. *arXiv preprint arXiv:2406.06382*, 2024.

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.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Vincent Tao Hu, David W Zhang, Yuki M Asano, Gertjan J Burghouts, and Cees GM Snoek. Self-guided diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18413–18422, 2023a.

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, 2023b.

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.

Amita Kamath, Cheng-Yu Hsieh, Kai-Wei Chang, and Ranjay Krishna. The hard positive truth about vision-language compositionality. In *European Conference on Computer Vision*, pp. 37–54. Springer, 2024.

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.

Aniket Kittur, Ed H Chi, and Bongwon Suh. Crowdsourcing user studies with mechanical turk. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pp. 453–456, 2008.

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.

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

Youwei Liang, Junfeng He, Gang Li, Peizhao Li, Arseniy Klimovskiy, Nicholas Carolan, Jiao Sun, Jordi Pont-Tuset, Sarah Young, Feng Yang, et al. Rich human feedback for text-to-image generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19401–19411, 2024.

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.

Chengjiang Long, Gang Hua, and Ashish Kapoor. Active visual recognition with expertise estimation in crowdsourcing. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3000–3007, 2013.

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.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

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

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35: 27730–27744, 2022.

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.

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.

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

Ugur Sahin, Hang Li, Qadeer Khan, Daniel Cremers, and Volker Tresp. Enhancing multimodal compositional reasoning of visual language models with generative negative mining. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 5563–5573, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Kunal Singh, Mukund Khanna, and Pradeep Moturi. Effective text-to-image alignment with quality aware pair ranking. In *NeurIPS 2024 Workshop on Fine-Tuning in Modern Machine Learning: Principles and Scalability*.

Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

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

Nishanth Upadhyaya and Raghavendra Sridharamurthy. Internalized self-correction for large language models. *arXiv preprint arXiv:2412.16653*, 2024.

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, 2023.

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.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 13484–13508, 2023.

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

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.

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, 2024a.

Kevin Yang, Dan Klein, Asli Celikyilmaz, Nanyun Peng, and Yuandong Tian. Rlcd: Reinforcement learning from contrastive distillation for lm alignment. In *The Twelfth International Conference on Learning Representations*, 2024b.

Shentao Yang, Tianqi Chen, and Mingyuan Zhou. A dense reward view on aligning text-to-image diffusion with preference. In *Forty-first International Conference on Machine Learning*.

Yongjin Yang, Sihyeon Kim, Hojung Jung, Sangmin Bae, SangMook Kim, Se-Young Yun, and Kimin Lee. Automated filtering of human feedback data for aligning text-to-image diffusion models. *arXiv preprint arXiv:2410.10166*, 2024c.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

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.

Yunan Zeng, Yan Huang, Jinjin Zhang, Zequn Jie, Zhenhua Chai, and Liang Wang. Investigating compositional challenges in vision-language models for visual grounding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14141–14151, 2024.

Chenhao Zheng, Jieyu Zhang, Aniruddha Kembhavi, and Ranjay Krishna. Iterated learning improves compositionality in large vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13785–13795, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

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.

A Failure Mode Atomization

A.1 Prompts used for analysing the model failures

This subsection lists the prompts used to analyze failure modes in visual text rendering. The prompts cover a wide range of visual contexts and word types, including rare, synthetic, and common terms, to evaluate textual accuracy under diverse conditions.

1. Black and white sign with the words ”activeware” on a white background, wireframe, generative art.
2. The slogan ”aftnerne” is written on the schoolbag.
3. Black and white sign with the words ”alvoni” on a white background, wireframe, generative art.
4. This car has a sign that reads ”amprofon”.
5. ”apazine” is written on the battery.
6. Little bee holding a sign that says ”bailup”.
7. In class, the teacher wrote the phrase ”brahmaic” on the blackboard.
8. At the airport, a sign that says ”burwardeslyn”.
9. In a hospital, a sign that says ”capitularis”.
10. A dark forest with only one light in the distance and the text ”chapultepee”.
11. In class, the teacher wrote the phrase ”chelonitis” on the blackboard.
12. This cosmetic bottle says ”churchilll”.
13. A lizard sitting on home plate at a baseball field with the words ”copden” in a speech bubble.
14. A cartoon of a cat with a thought bubble that says ”cubelo”.
15. There is a ”danakadan” sign in the hotel.
16. There is a notice board next to the train station that reads ”elitei”.
17. Small snail holding a sign that says ”ensco”.
18. At the train station, a sign that says ”galysheva”.
19. In the exhibition hall, a sign that reads ”gamengame”.
20. Minimal sculpture of the word ”garoff”, made of light metallic iridescent chrome thin lines, 3D rendering, isometric perspective, super detailed, dark background.
21. In the scenic spot, a sign that reads ”hallatar”.
22. There is a sign saying ”hammerclaw” in the mall.
23. The slogan ”harnetty” is posted on the bus stop.
24. ”heeresbekleidungsamt” slogan printed on school bus.
25. A lizard sitting on home plate at a baseball field with the words ”heyyyy” in a speech bubble.
26. In a supermarket, a sign that says ”josanas”.
27. Close up of a toothpaste tube figurine, 3D rendering, candy pastels, with the text ”kangun” on the tube.
28. A shot of a vine with the text ”kokachin” sprouting, centered.
29. The words ”kornephoros” are written on the paper towel.
30. Little frog holding a sign that says ”linneas”.
31. Pillow in the shape of ”malaefone”, Alphabetism, fun jumbled letters, [close-up] bread, graphic art.
32. ”managat” reminder posted on the bus.
33. Photos with ”mediashopping” sign.

34. A pumpkin with a beard, a monocle and a top hat with the text ”minioudaki” in a speech bubble.
35. Hubble and the Milky Way with the text ”monaut”.
36. Billboard with ”muthialpettah”.
37. A parrot on a pirate ship, a parrot wearing a pirate hat with the text ”odnoin”.
38. A plane flies over the city with the words ”ormelle” written in smoke trails.
39. The slogan of ”paikary” is written in the lottery station.
40. ”quercusglobulus” reminder posted on the bus.
41. A newspaper headline read ”qurniyya” and a photo showed a half-eaten pumpkin.
42. ”rimydis” reminder posted in the restaurant.
43. There is a notice in the supermarket that says ”saukrieg”.
44. Professionally designed logo for a bakery called ”scilliano”.
45. A globe with the words ”secciones” written in bold letters.
46. At the train station, a sign that says ”serranito”.
47. Professionally designed logo for a bakery called ”siyaya”.
48. In a hospital, a sign that says ”slevyas”.
49. Piggy holding a sign that says ”spissoy”.
50. Grow in a pretty pot with a ”sukhaybarat” sign.
51. Scrabble board showing the words ”taktai”.
52. A hastily handwritten note saying ”thangundi” posted on the fridge.
53. Photo of a helicopter with ”thellaru” written on the side, landing in a valley.
54. A shot of a vine with the text ”thornborg” sprouting, centered.
55. Billboard with ”unveiledness”.
56. ”vayalil” sign with home decor.
57. A street sign on the street reads ”wowomen”.
58. A t-shirt that says ”zwangsanstalt”.
59. There is a sign saying ”accept” in the mall.
60. Close up of a toothpaste tube figurine, 3D rendering, with the text ”assembly”.
61. The city of Toronto seen from the plane with the text ”away”.
62. A dark forest with only one light and the text ”boy”.
63. Little bee holding a sign that says ”cell”.
64. Professionally designed logo for a bakery called ”club”.
65. A photo of an aquarium with the words ”construct”.
66. A lizard sitting on home plate with the words ”debut” in a speech bubble.
67. There is a ”department” sign in the hotel.
68. Black and white sign with the words ”direct”, wireframe.
69. Professionally designed logo for a bakery called ”effort”.
70. Billboard with ”energy”.
71. A picture of the Earth with the words ”facility”.
72. A t-shirt that says ”feature”.
73. There is a notice in the supermarket that says ”financial”.
74. In a supermarket, a sign that says ”fly”.
75. There is a notice in the supermarket that says ”help”.
76. A street sign reads ”hospital”.

77. Scrabble board showing the words ”individual”.
78. At the train station, a sign that says ”lake”.
79. The words ”leave” are written on the paper towel.
80. A yellow saxophone in rainbow smoke with the words ”london”.
81. Black and white sign with the words ”look”, wireframe.
82. A detailed drawing with the text ”lord”.
83. Little frog holding a sign that says ”measure”.
84. Piggy holding a sign that says ”notable”.
85. ”october” generative art, sticky smoke made of dots.
86. Small snail holding a sign that says ”operate”.
87. ”party” reminder posted on the bus.
88. In class, the teacher wrote the phrase ”popular” on the blackboard.
89. At the train station, a sign that says ”professional”.
90. The slogan ”program” is posted on the bus stop.
91. Photo of a helicopter with ”public” written on the side.
92. The sign ”reach” hangs beside the swimming pool.
93. At the airport, a sign that says ”regular”.
94. Pillow in the shape of ”religious”, Alphabetism, graphic art.
95. In a hospital, a sign that says ”survive”.
96. A shot of a vine with the text ”teach” sprouting.

A.2 Prompts given to the expert (model) for the failure analysis

This subsection presents the prompts used to guide the expert model in analyzing text rendering failures in generated images. The prompts correspond to four stages of our analysis pipeline: (1) coarse failure mode identification (Figure 5), (2) atomization of failures into fine-grained visual factors (Figure 6), (3) filtering and consolidation of relevant failure modes (Figure 7), and (4) construction of visual transformation programs that replicate these failures (Figure 8). Together, these prompts enable systematic extraction and operationalization of structured failure patterns.

A.3 Coarse Failure Modes

This subsection lists some high-level failure categories identified during the initial analysis of generated images. These coarse failure modes capture common patterns of text rendering errors observed across samples.

1. **Failure Mode Name:** Misalignment of Text
   Description: The word “Popular” on the blackboard appears slightly misaligned with the center, resulting in imprecise placement.

2. **Failure Mode Name:** Misalignment of Characters
   Description: Many characters in the generated text are misaligned, creating a jumbled and disorganized appearance.

3. **Failure Mode Name:** Alignment Issue
   Description: The text appears unevenly distributed along the wall, showing poor centering and inconsistent spacing between characters.

4. **Failure Mode Name:** Alignment
   Description: The text “att1 W dvalie” appears misaligned, with uneven spacing and inconsistent placement relative to the background.

Figure 5: Prompt used to analyze generated images and identify coarse text-rendering failure modes present in the output.

⌨ Coarse Failure Mode Identification

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 6: Prompt used to decompose coarse failure modes into fine-grained, measurable visual factors that can be programmatically modeled.

⌨ Failure Mode Atomization

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., 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 7: Prompt used to refine and filter the extracted failure modes, removing redundancy and retaining only distinct, programmable categories.

⌨ 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]`

5. **Failure Mode Name:** Noise in Text
   Description: The text exhibits random distortions or visual noise, reducing clarity and legibility.

6. **Failure Mode Name:** Letter Spacing Inconsistency
   Description: The spacing between letters in "AFFENE" and "WECKES" varies noticeably, giving an unprofessional and uneven look.

7. **Failure Mode Name:** Illegible Text
   Description: The smaller text beneath "AFFENE" is blurry and difficult to read, with unclear characters that make it incomprehensible.

8. **Failure Mode Name:** Word-Level Inconsistency
   Description: The generated word "ALVIONI" is nonsensical and does not align with standard language conventions or the intended prompt.

A.4 Programmable Visual Cues

1. **Failure Mode:** Misleading Text Replication
   Fine-Grained Factors:
   - **Format Non-Adherence:** Text does not conform to standard license plate structures.
   - **Invalid Character Set:** Inclusion of symbols or letters not used in license plate conventions.
   - **Aspect Ratio Distortion:** Incorrect proportions of the text area relative to real license plate dimensions.
   - **Boundary Overflow:** Generated text exceeds the physical boundaries of the license plate.

2. **Failure Mode:** Misalignment of Text
   Fine-Grained Factors:

Figure 8: Prompt used to convert curated failure mode descriptions into executable visual transformation programs.

⌨ Visual Program Construction

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]`

- **Baseline Deviation:** Individual letters deviate from a horizontal baseline.
- **Vertical Offset Variance:** Inconsistent vertical positioning across characters.
- **Angular Skew:** Rotation or tilt of letters relative to the horizontal axis.
- **Spacing Misalignment:** Uneven vertical gaps between stacked letters or lines of text.
- **Non-Parallel Alignment to Object:** Text fails to align with the central axis or surface contours of the marker.

3. **Failure Mode:** Non-Standard Word Formation

   Fine-Grained Factors:

   - **Unrecognizable Word Structure:** Character sequences do not resemble meaningful or valid words.
   - **Phonetic Inconsistency:** Letter arrangements do not correspond to plausible phonetic patterns.
   - **Randomized Letter Placement:** Characters appear randomly ordered, violating linguistic norms.

4. **Failure Mode:** Glyph Merging

   Fine-Grained Factors:

   - **Character Overlap:** Adjacent letters (e.g., "P" and "T") intersect, forming ambiguous or merged shapes.
   - **Stroke Fusion:** Strokes of neighboring characters blend together due to shared pixel intensity.
   - **Proximity Deviation:** Letters are placed excessively close, increasing the likelihood of unintended merging.

B Visual Program Construction

B.1 Visual Transformation Programs

This subsection provides the implementation of visual transformation functions that replicate specific text rendering failures identified during the analysis stage (refer to Figures 9–23).

C Experimental Details

C.1 Dataset Properties

We collect 6,266 from Crello dataset comprises design templates with 37 properties organized into four categories:

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`.

Figure 9: Horizontal Misalignment Transformation

```python
import copy
import random

def horizontal_misalignment(example, transformation_parameters=None):
    """
    Introduces horizontal misalignment by deviating characters along the x-axis from their original positions.
    """
    if transformation_parameters is None:
        transformation_parameters = {
            "max_x_offset": 10  # Maximum horizontal deviation in pixels
        }

    new_example = copy.deepcopy(example)
    max_x_offset = transformation_parameters["max_x_offset"]

    for i, text in enumerate(new_example["text"]):
        if text:  # Check if there's text in the element
            text_length = len(text)
            x_offsets = [
                random.uniform(-max_x_offset, max_x_offset) for _ in range(text_length)
            ]
            new_example["x_offset"][i] = x_offsets  # Update x_offset map with random deviations

    return new_example
```

Figure 10: Vertical Misalignment Transformation

```python
def vertical_misalignment(example, transformation_parameters=None):
    """
    Introduces y-axis height variance across characters, causing uneven vertical alignment.
    """
    if transformation_parameters is None:
        transformation_parameters = {
            "max_offset": 10.0  # Maximum vertical offset for misalignment
        }

    new_example = copy.deepcopy(example)
    for i, element_type in enumerate(new_example["type"]):
        if element_type == 1:
            text_length = len(new_example["text"][i])
            max_offset = transformation_parameters["max_offset"]
            y_offsets = [random.uniform(-max_offset, max_offset) for _ in range(text_length)]
            new_example["y_offset"][i] = y_offsets

    return new_example
```

Figure 11: Angular Distortion Transformation

```python
def angular_distortion(example, transformation_parameters={"max_angle": 30}):
    """
    Introduces angular distortion to characters by rotating each character randomly within a range.
    """
    new_example = copy.deepcopy(example)
    max_angle = transformation_parameters["max_angle"]

    for i, text in enumerate(new_example["text"]):
        if text:  # Check if there's text in the element
            text_length = len(text)
            rotation_offsets = [
                random.uniform(-max_angle, max_angle) for _ in range(text_length)
            ]
            new_example["rotation_offset"][i] = rotation_offsets  # Update rotation_offset map with random deviations

    return new_example
```

Figure 12: Line Thickness Inconsistency Transformation

```python
def line_thickness_inconsistency(example, transformation_parameters={"max_variation": 3.0}):
    """
    Introduces variability in stroke width within or across characters
    to simulate inconsistency in line thickness.
    """
    new_example = copy.deepcopy(example)
    max_variation = transformation_parameters["max_variation"]

    for idx, text_element in enumerate(new_example["text"]):
        if new_example["type"][idx] == 1:
            # Add random thickness variation to each character
            text_length = len(new_example["text"][idx])
            thickness_variation = [
                random.uniform(-max_variation, max_variation) if random.choice([0, 1]) else 0
                for _ in range(text_length)
            ]

            new_example["font_bold"][idx] = thickness_variation  # Apply the variations

    return new_example
```

Figure 13: Aspect Ratio Distortion Transformation

```python
def aspect_ratio_distortion(example, transformation_parameters={"max_stretch": 1.5,
     "max_compression": 0.5}):
    """
    Distorts the aspect ratio of elements randomly to simulate stretching or compressing.
    """
    new_example = copy.deepcopy(example)
    max_stretch = transformation_parameters["max_stretch"]
    max_compression = transformation_parameters["max_compression"]

    for i in range(new_example["length"]):
        if new_example["type"][i] == 1:  # Apply transformation to text elements only
            aspect_ratio_factor = random.uniform(max_compression, max_stretch)
            new_example["width"][i] *= aspect_ratio_factor  # Stretch or compress width
            new_example["height"][i] /= aspect_ratio_factor  # Adjust height to maintain some
     balance

    return new_example
```

Figure 14: Text Content Mismatch Transformation

```python
def text_content_mismatch(example, transformation_parameters=None):
    """
    Introduces text mismatches by randomly substituting, omitting, or swapping characters in the text.
    """
    new_example = copy.deepcopy(example)
    text_elements_indices = [
        i for i, element_type in enumerate(new_example['type']) if element_type == 1]

    if not text_elements_indices:
        return new_example  # No text elements to modify

    for index in text_elements_indices:
        text = new_example['text'][index]
        if not text:
            continue

        # Randomly decide transformation type
        transformation_type = random.choice(['substitute', 'omit', 'swap'])

        # Text mismatch transformations
        if transformation_type == 'substitute':
            # Substitute random characters
            num_subs = random.randint(1, max(1, len(text) // 5))
            for _ in range(num_subs):
                pos = random.randint(0, len(text) - 1)
                new_char = random.choice('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789')
                text = text[:pos] + new_char + text[pos + 1:]

        elif transformation_type == 'omit':
            # Omit random characters
            num_omissions = random.randint(1, max(1, len(text) // 5))
            omit_indices = set(random.sample(range(len(text)), k=num_omissions))
            text = "".join(ch for i, ch in enumerate(text) if i not in omit_indices)

        elif transformation_type == 'swap':
            # Swap adjacent characters randomly
            num_swaps = random.randint(1, max(1, len(text) // 5))
            for _ in range(num_swaps):
                pos = random.randint(0, len(text) - 2)
                text = (
                    text[:pos]
                    + text[pos + 1]
                    + text[pos]
                    + text[pos + 2:]
                )

        # Update the text in the example
        new_example['text'][index] = text

    return new_example
```

Figure 15: Character Omission Transformation

```python
def character_omission(example, transformation_parameters={"max_omissions": 3}):
    """
    Simulates the failure mode of character omission by randomly removing characters
    from text elements.
    """
    new_example = copy.deepcopy(example)
    for i in range(len(new_example["text"])):
        text = new_example["text"][i]  # Get the text for each text element
        if text and len(text) > 1:  # Ensure text exists and is long enough
            num_omissions = random.randint(1, min(transformation_parameters["max_omissions"], len(text) // 2))
            omission_indices = set(random.sample(range(len(text)), k=num_omissions))
            new_text = "".join(ch for idx, ch in enumerate(text) if idx not in omission_indices)
            new_example["text"][i] = new_text  # Apply the transformed text
    return new_example
```

Figure 16: Unintended Glyph Addition Transformation

```python
def unintended_glyph_addition(example, transformation_parameters={"max_glyphs": 3,
    "glyphs_list": ["@", "#", "$", "%", "&", "*"]}):
    """
    Adds unintended glyphs or symbols to random positions in the text content.
    """
    new_example = copy.deepcopy(example)
    text_elements = new_example["text"]
    num_additions = random.randint(1, transformation_parameters["max_glyphs"])
    glyphs_to_add = random.choices(transformation_parameters["glyphs_list"], k=num_additions)

    for i, text in enumerate(text_elements):
        if text:  # Ensure the text is not empty
            positions = random.sample(range(len(text) + 1), k=num_additions)
            for pos, glyph in zip(positions, glyphs_to_add):
                text = text[:pos] + glyph + text[pos:]
            new_example["text"][i] = text

    return new_example
```

Figure 17: Resolution Drop Transformation

```python
def resolution_drop(example, transformation_parameters={"scale_factor_range": (0.4, 0.8)}):
    """
    Reduces the resolution of the rendered elements by scaling down and scaling back up,
    simulating low resolution or rendering artifacts.
    """
    new_example = copy.deepcopy(example)

    # Choose a random scale factor within the provided range
    scale_factor = random.uniform(*transformation_parameters["scale_factor_range"])

    # Adjust the canvas dimensions
    new_width = int(new_example["canvas_width"] * scale_factor)
    new_height = int(new_example["canvas_height"] * scale_factor)

    # Scale down the canvas dimensions
    new_example["canvas_width"] = new_width
    new_example["canvas_height"] = new_height

    # Apply scaling to individual elements
    for i in range(new_example["length"]):
        new_example["width"][i] *= scale_factor
        new_example["height"][i] *= scale_factor
        new_example["left"][i] *= scale_factor
        new_example["top"][i] *= scale_factor

    # Scale back to the original canvas size (introduces artifacts)
    scale_back_factor = 1 / scale_factor
    new_example["canvas_width"] *= scale_back_factor
    new_example["canvas_height"] *= scale_back_factor

    for i in range(new_example["length"]):
        new_example["width"][i] *= scale_back_factor
        new_example["height"][i] *= scale_back_factor
        new_example["left"][i] *= scale_back_factor
        new_example["top"][i] *= scale_back_factor

    return new_example
```

Figure 18: Scaling Distortion Transformation

```python
def scaling_distortion(example, transformation_parameters={"scale_min": 0.5, "scale_max": 1.5}):
    """
    Introduces random scaling distortion to elements on the canvas.
    """
    new_example = copy.deepcopy(example)
    scale_min = transformation_parameters["scale_min"]
    scale_max = transformation_parameters["scale_max"]

    for i in range(new_example["length"]):
        # Apply a random scaling factor to width and height of each element
        scale_factor = random.uniform(scale_min, scale_max)
        new_example["width"][i] *= scale_factor
        new_example["height"][i] *= scale_factor

        # Adjust `left` and `top` positions to simulate scaling distortions
        new_example["left"][i] += random.uniform(-5, 5) * scale_factor
        new_example["top"][i] += random.uniform(-5, 5) * scale_factor

    return new_example
```

Figure 19: Random Character Sequencing Transformation

```python
def random_character_sequencing(example, transformation_parameters={"enabled": True}):
    """
    Randomly shuffles characters in the text to generate gibberish text.
    """
    new_example = copy.deepcopy(example)
    if not transformation_parameters.get("enabled", True):
        return new_example

    for i in range(len(new_example["text"])):
        original_text = new_example["text"][i]
        if original_text:  # Ensure there's text to shuffle
            shuffled_text = ''.join(random.sample(original_text, len(original_text)))
            new_example["text"][i] = shuffled_text

    return new_example
```

Figure 20: Inconsistent Letter Capitalization Transformation

```python
def inconsistent_letter_capitalization(example, transformation_parameters={"max_capitalization_alterations": 5}):
    """
    Introduces random or irregular capitalization in text.
    """
    new_example = copy.deepcopy(example)
    text_elements = [i for i, t in enumerate(new_example["type"]) if t == 1]

    for idx in text_elements:
        text = new_example["text"][idx]
        num_alterations = random.randint(1, transformation_parameters["max_capitalization_alterations"])
        alteration_indices = random.sample(range(len(text)), k=min(num_alterations, len(text)))

        modified_text = ""
        for i, ch in enumerate(text):
            if i in alteration_indices:
                if ch.islower():
                    modified_text += ch.upper()
                elif ch.isupper():
                    modified_text += ch.lower()
                else:
                    modified_text += ch
            else:
                modified_text += ch

        new_example["text"][idx] = modified_text

    return new_example
```

Figure 21: Multilingual Confusion Transformation

```python
def multilingual_confusion(example, transformation_parameters={"scripts": ["Latin", "Cyrillic", "Arabic", "Devanagari"], "probability": 0.3}):
    """
    Introduces multilingual confusion by randomly replacing characters with characters from other scripts.
    """
    new_example = copy.deepcopy(example)
    if "text" not in new_example or not isinstance(new_example["text"], list):
        return new_example

    # Define character sets for different scripts
    script_characters = {
        "Latin": "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "Cyrillic": "                                                ",
        "Arabic": "                                          ",
        "Devanagari": "                                                  ",
    }

    available_characters = "".join(
        script_characters[script] for script in transformation_parameters["scripts"]
        if script in script_characters
    )

    for i, text in enumerate(new_example["text"]):
        if random.random() < transformation_parameters["probability"]:
            transformed_text = "".join(
                random.choice(available_characters) if random.random() < 0.5 else char
                for char in text
            )
            new_example["text"][i] = transformed_text

    return new_example
```

Figure 22: Font Style Inconsistency Transformation

```python
def font_style_inconsistency(example, transformation_parameters={"max_variations": 3}):
    """
    Introduces font style inconsistencies within the same text block.
    """
    new_example = copy.deepcopy(example)
    max_variations = transformation_parameters["max_variations"]

    for idx, text in enumerate(new_example["text"]):
        if new_example["type"][idx] != 1:
            continue

        text_length = len(text)
        num_variations = random.randint(1, min(max_variations, text_length // 2))
        variation_indices = random.sample(range(text_length), k=num_variations)
        variation_indices.sort()

        for i in variation_indices:
            if new_example["font_bold"][idx]:
                new_example["font_bold"][idx][i] = random.choice([True, False])
            if new_example["font_italic"][idx]:
                new_example["font_italic"][idx][i] = random.choice([True, False])
            if new_example["font_size"][idx]:
                new_example["font_size"][idx] *= random.uniform(0.8, 1.2)
            if new_example["text_color"][idx]:
                color = new_example["text_color"][idx][i]
                r, g, b, a = map(float, color.replace("rgba(", "").replace(")", "").split(","))
                new_example["text_color"][idx][i] = f"rgba({r + random.uniform(-10, 10)}, {g + random.uniform(-10, 10)}, {b + random.uniform(-10, 10)}, {a})"

    return new_example
```

Figure 23: Readability Deviation in Small Text Transformation

```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.
    """
    new_example = copy.deepcopy(example)

    for i, element_type in enumerate(new_example["type"]):
        if element_type == 1:
            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

            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

            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 24: Kerning and Spacing Irregularity Transformation

```python
def kerning_and_spacing_irregularity(example, transformation_parameters=None):
    """
    Introduces uneven spacing between characters or words.
    """
    new_example = copy.deepcopy(example)
    transformation_parameters = transformation_parameters or {"max_offset": 5.0, "irregularity_probability": 0.5}

    for i in range(new_example["length"]):
        if new_example["type"][i] == 1:
            text_length = len(new_example["text"][i])

            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
            ]
            new_example["letter_spacing"][i] = sum(letter_spacing_values) / len(letter_spacing_values)

    return new_example
```

Figure 25: Word Break Errors Transformation

```python
def word_break_errors(example, transformation_parameters={"break_probability": 0.5,
    "max_fragments": 3}):
    """
    Introduces word break errors by splitting words improperly.
    """
    new_example = copy.deepcopy(example)
    text = new_example["text"]

    for i, line in enumerate(text):
        words = line.split()
        transformed_line = []
        for word in words:
            if random.random() < transformation_parameters["break_probability"]:
                break_count = random.randint(1, transformation_parameters["max_fragments"])
                break_points = sorted(random.sample(range(1, len(word)), k=min(break_count,
    len(word)-1)))
                fragments = []
                start = 0
                for point in break_points:
                    fragments.append(word[start:point])
                    start = point
                fragments.append(word[start:])
                transformed_line.extend(fragments)
            else:
                transformed_line.append(word)

        new_example["text"][i] = " ".join(transformed_line)

    return new_example
```

C.2 Prompt used For OCR based 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 any type of captioning.*

Important Note:
I am going to use your response in evaluation for the images, so I expect you to return the exact text that you are seeing in the image without any correction to have a fair comparison.

D Qualitative Results

The qualitative comparisons in Tables 5 and 6 present visual differences across model variants. Across the examples, we observe improved text rendering and better consistency in generated outputs. These trends are consistently visible in both Diffusion-DPO and DSPO based configurations. Overall, the qualitative results highlight clearer and more reliable visual generation.

E Dataset Comparison: Model-Generated vs. F-GPS Framework

We compare two approaches for generating negative samples in our preference dataset: (1) model-generated samples directly from text-to-image models, and (2) synthetically constructed samples using our F-GPS framework. Both approaches produce preference pairs consisting of winning images (higher quality) and losing images (lower quality) for identical text prompts. Table 7 presents examples from the model-generated approach, while Table 8 shows corresponding examples from

Table 5: Qualitative comparison of SDXL-Base, SFT and Diffusion-DPO based different configurations.

SDXL-Base	SFT	DPO	DPO with Shuffling	DPO with F-GPS

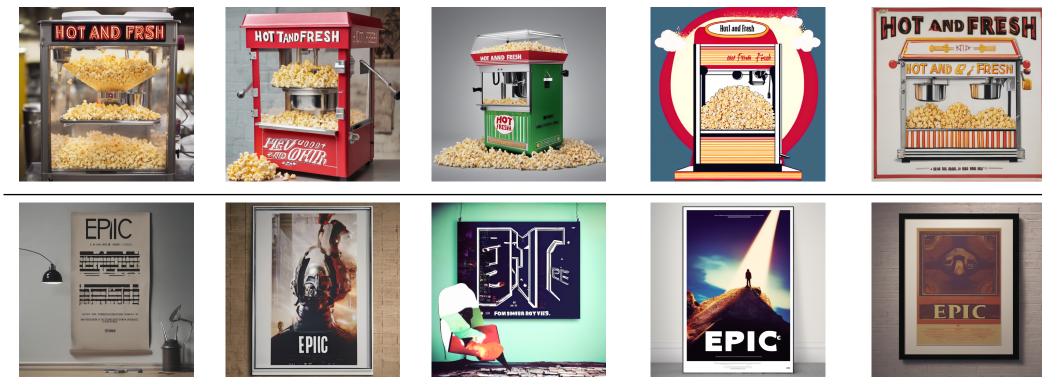
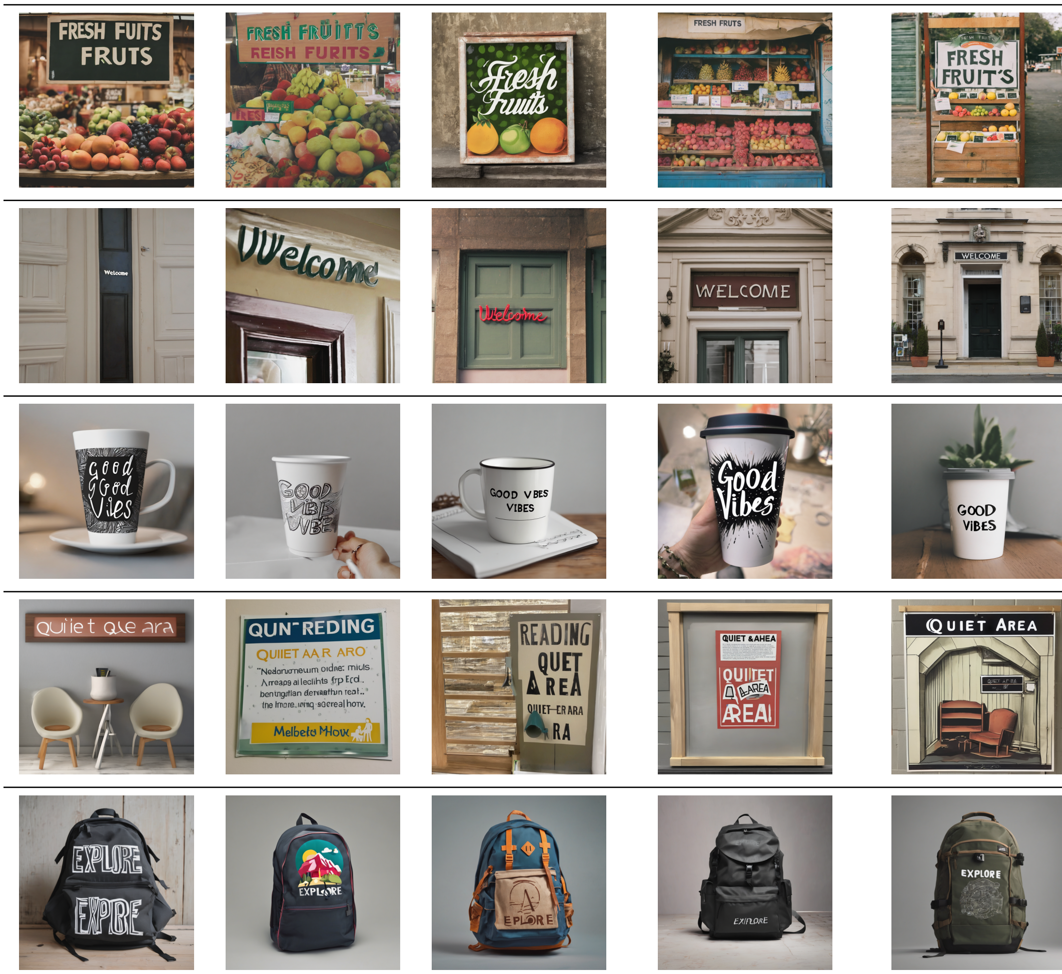


Table 6: Qualitative comparison of SDXL-Base, SFT and DSPO based different configurations.

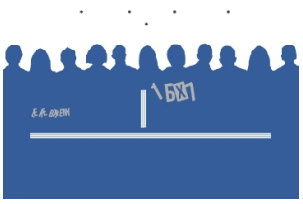
SDXL-Base	SFT	DSPO	DSPO with Shuffling	DSPO with F-GPS



the F-GPS framework. Each table contains four diverse prompts spanning different domains and visual requirements, along with their respective preference pairs.

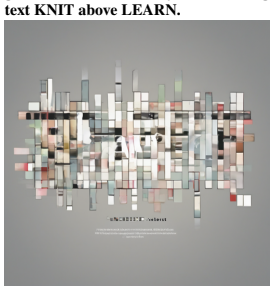
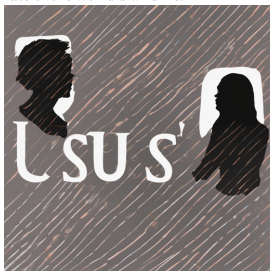
The key distinction between these approaches lies in the controllability and interpretability of failure modes. Model-generated negative samples exhibit organic but unpredictable failure patterns that arise naturally from the generation process. In contrast, F-GPS-generated samples incorporate specific, programmable failure modes such as horizontal misalignment, vertical offset variance, angular distortion, and character-level inconsistencies. This systematic approach enables targeted analysis of model weaknesses and facilitates more effective preference learning through structured negative examples.

Table 7: Preference pairs with F-GPS-generated negatives. For each prompt, a preferred (winning) image is paired with a losing image synthesized using controlled visual programs, enabling interpretable and targeted failure modeling.

Prompt	**Winning Image**	**Losing Image**
Generate an image of an architect working with blueprints; display text 'building' and 'core' vertically.		
Generate an image with background as garage sale scene; text '15TH JUNE 10 AM TO 5 PM ZAINAB'S CHARITY GARAGE SALE CLOTHING & HOME STUFF' written on it.		
Generate an image with background as Berkeley startup summit scene; text '2017' then 'IN BERKELEY' written on it.		
Generate an image with background as origami paper texture and text 'LEARN GREAT TIPS ON MAKING FLOWERS AND ANIMALS! ORIGAMI CLASS' written on it.		

The comparison reveals several advantages of the F-GPS framework. First, the synthetic negative samples exhibit precisely controlled failure modes that can be systematically varied in intensity and type (see Table 8). Second, the framework enables generation of negative samples for any

Table 8: Negative samples constructed from SDXL-generated outputs, where failures emerge organically without explicit control.

Generate an image with a teddy bear in a baby car seat; overlay text: Molly's Baby Car Seats, You Baby Is Safe Now. 	**Generate an image with a blue background and needle and yarn, featuring text KNIT above LEARN.** 
Generate an image with festive red background and text Happy New Year then the year of the dog. 	**Generate an image with a spa salon theme, featuring plant sketches and text $200 GIFT VOUCHER vertically centered.** 
Generate an image with a birthday party in South Ozone Park; text MY BIRTHDAY TODAY IS written vertically. 	**Generate an image of a beautiful young girl in sunglasses, with text by Jameson J. and Tampa Street Style Lookbook printed on it.** 
Generate an image with candles in frame and text KENNER BAPTIST CHURCH Join us this weekend for the 60th Anniversary Service. 	**Generate an image with a nature drawing background and text The action taken by each of us will determine the fate of the world's wildlife.** 

arbitrary prompt without requiring additional model inference. Third, the programmatic nature of F-GPS allows for comprehensive coverage of failure mode combinations, ensuring that the preference dataset captures diverse degradation patterns. These characteristics make F-GPS-generated samples particularly valuable for training robust preference models that can identify and penalize specific text rendering failures.