FAILURE-AWARE ADAPTIVE RECOVERY FOR TEXT-TO-IMAGE GENERATION: WHEN TO RESAMPLE AND WHEN TO REWRITE

Anonymous authors
Paper under double-blind review

ABSTRACT

A failed text-to-image generation reveals what went wrong, but not how additional inference compute should be used for recovery. We show that post-failure recoverability is intervention-dependent: resampling the original prompt provides a strong baseline, while prompt rewriting recovers a complementary subset of failures missed by seed exploration. This motivates viewing recovery as a budget-allocation problem over stochastic and semantic interventions rather than as uniform prompt refinement. We introduce FARE, a failure-aware inference-time framework that uses evidence from the initial failed generation to guide both recovery allocation and early trajectory pruning. FARE estimates the recovery utility of feasible prompt–seed portfolios and conservatively retains resampling unless an alternative allocation provides sufficient predicted benefit. It then explores the selected candidates in parallel and uses the observed failed trajectory as a request-specific reference to prune candidates likely to reproduce the same failure. Across multiple T2I benchmarks, FARE improves recovery over fixed recovery policies under matched post-failure budgets, and its advantage persists as the available inference budget increases. Across SD2.1, SDXL, and FLUX.1-dev, failure-aware allocation improves recovery over the train-selected fixed allocation by 3.6–4.6 percentage points. We further find that matched failure references provide stronger early recovery signals than shuffled references, enabling conservative pruning that reduces redundant denoising while largely preserving recovery. These results recast post-failure prompt optimization as a failure-conditioned inference-time search problem, in which prompt rewriting is one intervention alongside stochastic exploration rather than the default response to generation failure.

1 INTRODUCTION

Text-to-image (T2I) generation has advanced rapidly, with latent diffusion and rectified-flow models enabling high-quality image synthesis from natural-language descriptions (Rombach et al., 2022; Esser et al., 2024). However, visual realism does not necessarily imply faithful realization of compositional requirements. Fine-grained evaluations continue to reveal difficulties with object counts, attribute binding, spatial relations, and combinations of multiple semantic constraints (Hu et al., 2023; Kamath et al., 2025). Even when a prompt is explicit, a generated image may omit a required object, reverse a relation, or attach an attribute to the wrong entity. These failures motivate a practical inference-time question: once a generation is known to be semantically incorrect, how should additional computation be used to recover it?

Recent work suggests two broad ways of spending additional inference compute. One direction modifies the textual conditioning using prompt optimization, visual feedback, or iterative refinement (Wu et al., 2026; Chen et al., 2025; Liu et al., 2026; Jiang et al., 2026). Another exploits the stochasticity of generative models by sampling alternative noise seeds, ranking multiple candidates, or progressively pruning weak trajectories (Zhuo et al., 2025; Guimaraes & Perona, 2026; Huang et al., 2026; Guo et al., 2026). These studies establish that both semantic intervention and stochastic exploration can improve T2I generation. However, they also expose a missing decision problem:

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.

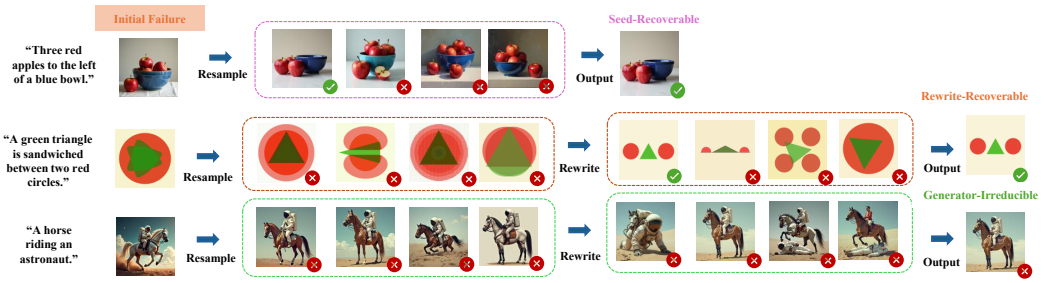


Figure 1: **Illustration of intervention-dependent post-failure recoverability.** After an initial semantic failure, different requests exhibit distinct recovery behaviors. Some failures are corrected by resampling the original prompt with a new seed (seed-recoverable), while others are recovered only after prompt rewriting (rewrite-recoverable); some remain unsuccessful under both interventions within the evaluated budget (unrecovered under budget). Examples are generated with FLUX.1-dev.

after observing a particular failed generation, it is generally unclear which intervention is worth the next unit of inference compute.

Our empirical results reveal budget-dependent and complementary recovery behavior. On T2I-CompBench with Stable Diffusion 2.1, resampling recovers 70.7% (816/1154) of initial failures within nine additional samples, showing that prompt modification is often unnecessary. Under matched budgets, resampling outperforms rewriting with three generations (45.2% vs. 43.4%), but rewriting performs better with nine (74.7% vs. 70.7%). Their union reaches 80.2% under a larger combined budget, revealing complementary coverage rather than a matched-compute gain. These findings motivate using initial-failure evidence to allocate a fixed recovery budget between resampling and rewriting.

These observations motivate characterizing failures by their recoverability under different interventions, rather than attributing each failure to a unique latent cause such as "sampling error" or "prompt error." A single failed generation does not establish whether resampling, rewriting, or neither will succeed within the remaining budget. We therefore formulate recovery as a failure-conditioned inference-time allocation problem: using the available failure evidence to estimate the recovery utility of alternative allocations and distribute a fixed budget across original-prompt resampling and semantics-preserving rewrites.

Based on this formulation, we propose FARE, a failure-aware adaptive recovery framework. FARE uses structured initial-failure evidence and rewrite-side information to predict the recovery probability of feasible allocations across original-prompt resampling and semantics-preserving rewrite branches. A conservative policy retains an all-resampling allocation unless an alternative offers a sufficiently large predicted gain. The selected candidates are then generated in parallel and verified against the original user prompt, so rewriting changes only the search conditioning, not the evaluation target.

Allocation determines where to search, while early pruning determines which candidates continue receiving compute. Prior work shows that intermediate denoiser activations, cross-attention patterns, and partially denoised estimates can support early candidate assessment (Huang et al., 2026; Guo et al., 2026; Guimaraes & Perona, 2026). We investigate whether the initial failed trajectory provides an additional request-specific reference for recovery. FARE compares candidate trajectories with this reference and conservatively prunes candidates whose similarity remains high and stable across early checkpoints. Together, allocation and pruning use the initial failure to guide both the distribution and the realized consumption of recovery compute.

Our main contributions are summarized as follows:

- We show that seed resampling and prompt rewriting recover overlapping but distinct subsets of failed requests under matched post-failure budgets. This motivates formulating recovery as an adaptive search problem over seed and prompt spaces rather than relying on a single intervention.
- We propose FARE, which uses structured evidence from the initial failed generation to conservatively allocate a fixed recovery budget across original-prompt resampling and

failure-conditioned rewrite operators. FARE further uses the observed failed trajectory as a request-specific reference to prune unpromising candidates during early denoising.

- Extensive experiments demonstrate that FARE achieves strong semantic recovery under fixed inference budgets while reducing redundant denoising with only a small loss in recovery. Matched failure references also provide stronger early recovery signals than shuffled references and preserve more recoverable requests than generic pruning at matched generation cost.

2 Related Work

Failure-aware refinement and inference-time scaling. Generated images can provide useful feedback for improving text-to-image generation after an initial failure. VisualPrompter (Wu et al., 2026) identifies unsatisfied semantic concepts and uses them to guide targeted prompt refinement, while T2I-Copilot (Chen et al., 2025) integrates prompt interpretation, image evaluation, and iterative regeneration in a closed-loop pipeline. TARA (Liu et al., 2026) further decomposes semantic failures and routes failed propositions to type-specific repair strategies. These methods primarily improve recovery by adapting the textual conditioning based on visual feedback. A complementary line of work studies inference-time scaling by spending additional computation on multiple candidates or intervention spaces. ReflectionFlow (Zhuo et al., 2025) explores noise-, prompt-, and reflection-level scaling, while RAISE (Jiang et al., 2026) combines resampling, prompt rewriting, and instructional editing within an adaptive refinement procedure. Together, these works demonstrate the benefit of combining semantic refinement with broader inference-time search. However, how to distribute a fixed post-failure budget across different recovery mechanisms remains less explored. FARE focuses on this allocation problem using evidence from the initial failed generation.

Early trajectory assessment and candidate pruning. A separate line of work reduces inference cost by predicting candidate quality before generation is complete. Diffusion Probe (Huang et al., 2026) uses intermediate cross-attention representations to predict final image quality, while ProbeSelect (Guo et al., 2026) exploits intermediate denoiser activations to identify promising seeds. Progressive Seed Pruning (Guimaraes & Perona, 2026) evaluates partially denoised candidates and progressively removes low-potential trajectories under a fixed model-evaluation budget. For instruction-guided editing, ELECT (Kim et al., 2025) uses early latent information to assess candidate quality before full generation. Prior approaches largely assess each partial trajectory using generic quality or task-specific signals. In post-failure recovery, the observed failed trajectory provides an additional request-specific reference, which FARE uses to identify candidates likely to reproduce the same failure.

3 Method

We formulate text-to-image failure recovery as a **failure-conditioned inference-time allocation problem**. Given an initial generation that violates the semantic requirements of the user prompt, our goal is to estimate the expected recovery utility of alternative inference-time interventions and allocate a finite post-failure budget accordingly.

Our framework consists of three stages. First, we diagnose the initial failed generation and extract structured failure evidence describing the violated semantic constraints. Second, conditioned on this evidence, FARE predicts the recovery utility of feasible budget allocations over original-prompt resampling and failure-conditioned rewrite operators, while retaining a strong resampling default when adaptation is uncertain. Third, the selected recovery candidates are explored in parallel, and a failure-referenced early probe compares their intermediate trajectories with the initial failure to terminate candidates unlikely to recover. FARE makes two complementary inference-time decisions: a request-level decision of where to allocate recovery compute, followed by a trajectory-level decision of which allocated candidates should continue receiving compute.

3.1 Problem Formulation and Failure Evidence

Let p denote the original user prompt and G a frozen text-to-image generator. We first sample $I_0 = G(p, s_0)$ using an initial seed s_0. A semantic verifier V evaluates whether I_0 satisfies the

requirements specified by p. Successful generations terminate immediately; otherwise, the failed sample becomes the observation from which subsequent recovery decisions are made.

To obtain structured failure evidence, a vision-language analyzer D decomposes the prompt into atomic semantic constraints and returns the violated subset $e_0 = D(p, I_0)$, including errors in object existence, counting, attribute binding, spatial relations, and actions. We summarize the information available before any recovery generation as $E_0 = (p, I_0, e_0)$. The central question is therefore: given only E_0 and a finite post-failure inference budget B, how should this budget be distributed between stochastic seed exploration and semantic prompt intervention? The post-failure recovery problem is then to use E_0 to allocate a finite inference budget B among feasible recovery interventions, while all resulting candidates are ultimately evaluated against the original prompt p.

3.2 Failure-Aware Budget Allocation

After the initial generation fails, FARE must decide how to spend the remaining post-failure budget. We consider four recovery branches: resampling the original prompt and three failure-conditioned rewrite operators, denoted by o_1, o_2, and o_3. An allocation is therefore a tuple $a = (b_0, b_1, b_2, b_3)$, where b_0 denotes the number of original-prompt resamples and $b_{1:3}$ denote the numbers of samples assigned to the three rewrite operators, subject to $\sum_{j=0}^{3} b_j = B$. For the default budget $B = 9$, this yields 220 possible allocations before operator availability is considered. For each failed request i, we retain only the feasible subset $\mathcal{A}_i(B)$ whose assigned rewrite operators pass their corresponding availability gates. Operator definitions and availability gates are detailed in Appendix B.

Counterfactual recovery supervision. For every failed request and feasible allocation a, we construct a binary counterfactual label indicating whether that allocation would recover the request. Let $C_i(a)$ denote the set of at most B candidates generated according to a. We define

$$Y_i(a) = \mathbb{I}\left[\max_{c \in C_i(a)} S(p_i, c) \geq \tau_{\text{succ}}\right],$$

where S is the semantic verifier score against the *original* prompt and τ_{succ} is the success threshold (0.5 for Soft-TIFA GM in our implementation). Thus, $Y_i(a) = 1$ if at least one candidate contained in allocation a successfully recovers the original request. Each request–allocation pair forms one training example. Its features combine three sources of information: (i) initial-failure evidence $E_{0,i}$, such as the number and types of violated semantic constraints; (ii) the allocation itself, including how the budget is distributed across resampling and the three rewrite operators; and (iii) rewrite-side information, including operator availability and the failed constraints targeted by each rewrite. We progressively enrich these features from F_0 to F_2, with an additional variant incorporating the text embedding of the original prompt. A shared predictor learns

$$q_\theta(i, a) = P(Y_i(a) = 1 \mid E_{0,i}, a, R_i),$$

where R_i summarizes the available rewrite information. All allocations belonging to the same request are kept in the same data split to prevent counterfactual leakage across training and evaluation.

Allocation at inference time. For a new failed request, FARE first determines which rewrite operators are available, enumerates the corresponding feasible allocation set $\mathcal{A}_i(B)$, and predicts a recovery probability for each allocation. The highest-scoring alternative is

$$\tilde{a}_i = \arg\max_{a \in \mathcal{A}_i(B)} q_\theta(i, a).$$

To avoid unnecessary deviations from a strong resampling baseline, we use a conservative selection rule. Let $a_0 = (B, 0, 0, 0)$ denote always-resample. We choose $\tilde{a}_i$ only when its predicted utility exceeds that of a_0 by at least a margin δ; otherwise we fall back to a_0:

$$a_i^\star = \begin{cases} \tilde{a}_i, & q_\theta(i, \tilde{a}_i) - q_\theta(i, a_0) \geq \delta, \\ a_0, & \text{otherwise.} \end{cases}$$

We use $\delta = 0.02$ unless otherwise specified. The predictor itself does not generate images; it only outputs the selected allocation $a_i^\star$. The corresponding candidates are then generated in parallel and processed by the early-pruning procedure described in Sec. 3.3. Additional details on allocation labels, predictor features, training, and the conservative policy are provided in Appendix C.

Algorithm 1 Failure-Aware Adaptive Recovery (FARE)

Require: Prompt p, generator G, verifier V, budget B, allocation predictor q_θ, early checkpoints $\mathcal{T}_{\text{early}}$, pruning thresholds $\tau_{\text{sim}}, \tau_U$

Generate initial image $I_0 = G(p, s_0)$ and retain its early trajectory
if $V(p, I_0) = 1$ **then**
  return I_0
end if
$e_0 \leftarrow D(p, I_0)$, $E_0 \leftarrow (p, I_0, e_0)$ *// failure evidence*
Determine available rewrite operators o_1 (local), o_2 (reinforce), and o_3 (restructure)
Construct feasible allocations $\mathcal{A}_i(B) = \{a = (b_0, b_1, b_2, b_3) : \sum_j b_j = B,\ b_j = 0 \text{ if } o_j \text{ is unavailable}\}$
$\tilde{a}_i \leftarrow \arg\max_{a \in \mathcal{A}_i(B)} q_\theta(E_0, a)$
$a_i^\star \leftarrow \tilde{a}_i$ if $q_\theta(E_0, \tilde{a}_i) - q_\theta(E_0, a_0) \geq \delta$, else $a_0 = (B, 0, 0, 0)$ *// failure-aware allocation*
Launch recovery candidates $\mathcal{C}_i(a_i^\star)$ in parallel
for each active candidate $c \in \mathcal{C}_i(a_i^\star)$ **do**
  for each early checkpoint $t \in \mathcal{T}_{\text{early}}$ **do**
    Measure failure-reference similarities $\big(s_t^{\text{dir}}(c),\ s_t^{\text{attn}}(c)\big)$ relative to the seed-0 failed trajectory
  end for
  Aggregate checkpoint-wise similarities into failure-reference score $S(c)$
  Compute temporal-consistency statistic $U(c)$
  if $S(c) > \tau_{\text{sim}}$ **and** $U(c) < \tau_U$ **and** more than one candidate remains active **then**
    prune c
  end if
end for *// conservative failure-referenced pruning*
Verify completed candidates against the original prompt p
return the highest-scoring successful candidate, or recovery failure

3.3 Failure-Referenced Early Trajectory Pruning

The selected allocation determines where recovery compute is spent, but fully denoising every candidate remains wasteful. We therefore use the initial failed trajectory as a request-specific reference and identify candidates that begin to reproduce the same failure during early denoising. Our design is motivated by the high-noise semantic transport analysis of ChordEdit Li et al. (2026); Lu et al. (2026), which shows that semantically meaningful generation directions already emerge in the early, high-noise regime. Following this insight, we inspect an early trajectory window around $t \in [0.75, 0.9]$, where trajectory information is already informative while substantial generation compute can still be saved.

Failure-referenced trajectory features. For each recovery candidate c, we compare its early trajectory with the corresponding seed-0 failed trajectory at several checkpoints using two complementary signals. First, we extract the classifier-free guidance direction $g_t^c = \epsilon_{\text{cond}}(x_t^c, t) - \epsilon_{\text{uncond}}(x_t^c, t)$ and compress it into a coarse energy representation $\mathcal{E}_4(g_t^c)$. We then measure the similarity of both the generation direction and the failed-atom attention pattern:

$$s_t^{\text{dir}}(c) = \cos\big(\mathcal{E}_4(g_t^c), \mathcal{E}_4(g_t^0)\big)\,,$$
$$s_t^{\text{attn}}(c, r) = \cos\big(A_t^c(r), A_t^0(r)\big)\,, \qquad r \in e_0.$$

The first signal measures whether the candidate follows a coarse semantic transport pattern similar to the trajectory that previously failed. The second focuses specifically on the semantic atoms violated in the initial image, comparing their coarse cross-attention patterns with those of the failed reference. Thus, the trajectory comparison captures both global similarity to the failed generation process and similarity around the semantic structures that were actually incorrect. Detailed feature construction, spatial pooling, and checkpoint aggregation are provided in Appendix D.

Conservative failure-referenced pruning. A high failure-reference similarity suggests that a candidate is beginning to reproduce the observed failure, but an isolated similarity peak may be caused by transient early noise. We therefore aggregate the checkpoint-wise evidence into a failure-reference

score $S(c)$ and require the similarity to remain stable across the observed early trajectory. Let $U(c)$ denote the cross-checkpoint variation of the similarity signal. We prune a candidate only when

$$S(c) > \tau_{\text{sim}} \quad \text{and} \quad U(c) < \tau_U,$$

where the thresholds are selected on validation requests. This consistency constraint makes pruning deliberately conservative: a candidate is terminated only when it remains strongly and persistently similar to the request-specific failed trajectory. We additionally retain at least one candidate for each request; all surviving candidates continue to full denoising and are evaluated against the original prompt. Details of threshold selection and temporal consistency are provided in Appendix D.

4 Experiments

4.1 Experimental Setup

Benchmarks and generation. We evaluate FARE on GenEval2 (Kamath et al., 2025), T2I-CompBench (Huang et al., 2023) (hereafter CompBench), and TIFA (Hu et al., 2023) using Stable Diffusion 2.1. For each benchmark, we first construct an initial-failure pool by retaining requests whose seed-0 generation fails the semantic verifier. Different analyses use fixed subsets of this pool according to their experimental purpose: Sec. 4.2 reports descriptive recoverability over the full analysis pool, whereas learned recovery policies in Sec. 4.3 are evaluated on held-out requests after request-level train/validation/test splitting. Scaling and pruning experiments use their own fixed evaluation pools, shared by all methods compared within each experiment. Unless otherwise specified, each image uses $T = 50$ denoising steps and the post-failure budget is $B = 9$ full-generation equivalents. More details are provided in Appendix A.1–A.3

Evaluation. All recovery candidates, including those generated from rewritten prompts, are evaluated against the original user prompt. We report request-level recovery, where a request is successful if at least one completed candidate satisfies all semantic constraints, and generation cost, measured by the executed denoising steps normalized by one full generation. Early-terminated trajectories contribute only the computation already executed, and the shared initial generation is excluded from the post-failure budget. Formal metric definitions and compute-accounting details are provided in Appendix A.4.

Protocol. We compare fixed resampling and rewriting strategies, mixed recovery portfolios, existing adaptive refinement and inference-time scaling methods, and FARE under matched post-failure budgets. All candidate outcomes associated with the same original request are kept in the same data split, and allocation models and pruning operating points are selected using training and validation data only. Additional protocol details are provided in Appendix A.4–A.5.

4.2 Heterogeneous Recoverability after Initial Failure

We first examine whether resampling and prompt rewriting recover the same initial failures under matched post-failure budgets. We evaluate the 200 prompts in DrawBench (Saharia et al., 2022), a stratified 600-prompt subset of TIFA (TIFA-600), and 2400 examples from CompBench. Figure 2 partitions each failure into four regimes: seed-only, rewrite-only, both, and neither, according to whether recovery is achieved by additional seeds, rewritten prompts, both interventions, or neither.

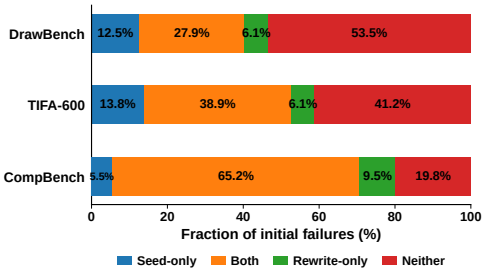


Figure 2: **Post-failure recoverability on the full analysis pool.**

The two interventions exhibit substantial but incomplete overlap. On DrawBench, resampling recovers 40.4% of initial failures and rewriting recovers 34.0%, while their union reaches 46.5%. A similar trend appears on TIFA-600, where the corresponding recovery rates are 52.7%, 45.0%, and 58.8%. In contrast, on CompBench, rewriting recovers 74.7% of initial failures, slightly exceeding the 70.7% recovered by resampling, while their union further increases recovery to 80.2%.

Table 1: Post-failure recovery under a maximum budget of $B = 9$ additional generations using SD 2.1 with 50 denoising steps. Recovery is evaluated on the shared initial-failure subset. VLM Calls counts vision-language model invocations used internally during recovery, excluding the shared final evaluator. DSteps denotes the realized number of diffusion denoising steps executed after the initial failure, including steps from early-terminated trajectories.

	Recovery (%) ↑			**VQA ↑**	**Efficiency ↓**	
Method	GenEval2	TIFA	CompBench	Avg.	VLM Calls	Avg. DSteps
Fixed Recovery Policies						
Always Resample	12.7	24.2	74.5	0.2180	0	450
Always Rewrite	11.3	23.6	74.0	0.2018	1	450
Train-selected Fixed	10.2	25.2	75.3	0.2238	1	450
Feedback-based Recovery						
VisualPrompter (Wu et al., 2026)	12.4	24.8	75.1	0.2157	≥ 9	450
TARA (Liu et al., 2026)	13.1	25.3	75.8	0.2214	2	450
PRIS (Kim et al., 2026)	13.4	**25.5**	76.0	0.2228	≥ 9	450
T2I-Copilot (Chen et al., 2025)	12.9	25.0	75.6	0.2204	≥ 19	≤ 450
RAISE (Jiang et al., 2026)	13.2	25.2	75.9	0.2217	3	400
Inference-time Scaling						
Noise Scaling (Ma et al., 2025)	13.0	24.7	75.5	0.2200	0	450
Noise/Prompt Scaling (Zhuo et al., 2025)	13.3	25.1	75.9	0.2221	9	450
Ours						
FARE w/o pruning	**13.8**	25.4	**76.2**	**0.2243**	1	450
FARE w/ pruning	13.5	25.2	75.8	0.2224	1	**350**

Rewrite-only cases account for 6.1%, 6.1%, and 9.5% of failures across DrawBench, TIFA-600, and CompBench, respectively, while a distinct subset remains recoverable only through seed exploration.

These results establish two properties of post-failure recovery. First, neither intervention consistently dominates across benchmarks: resampling is stronger on DrawBench and TIFA-600, whereas rewriting is stronger on CompBench. Second, the two interventions provide complementary recovery, as each recovers failures that the other misses. Thus, committing the entire post-failure budget to either seed or prompt exploration is suboptimal for a subset of requests, motivating failure-conditioned allocation of inference compute across the two recovery spaces. Rewrite operators and their semantic-preservation constraints detailed in Appendix B.

4.3 Failure-Aware Recovery under a Common Recovery Budget

We evaluate whether evidence from the initial failed generation can improve recovery under a common post-failure generation budget. All methods operate on the same initial-failure sets with a maximum budget of $B = 9$ additional generations. With SD 2.1 and $T = 50$ denoising steps, a method that exhausts the full generation budget incurs 450 denoising steps. The budget is a maximum rather than a requirement: adaptive stopping and trajectory pruning may reduce the realized generation cost. FARE without pruning is the primary comparison for isolating the effect of failure-aware allocation, while the pruned variant reports the additional compute reduction obtained from early trajectory termination; pruning is analyzed in detail in Sec. 4.5.

Table 1 compares three classes of baselines: fixed recovery policies, feedback-based recovery methods, and inference-time scaling approaches. Fixed policies use the same allocation for every failed request, whereas feedback-based and scaling methods adapt the conditioning or search process using intermediate feedback. FARE instead uses the initial-failure evidence to select among feasible allocations over original-prompt resampling and the three failure-conditioned rewrite operators, while retaining resampling as a conservative default. Details of the feasible allocation space, counterfactual supervision, predictor features and training, and conservative selection policy are provided in Appendix C.

Recovery performance. No single fixed recovery policy is uniformly strongest across benchmarks. Among the fixed policies, Always Resample performs best on GenEval2, whereas Train-selected Fixed is stronger on TIFA and CompBench, indicating that the utility of a recovery portfolio varies

Table 2: **Generalization across generators.** GenEval2 recovery under $B = 9$. Parentheses show FARE's gain over Fixed.

	Recovery (%) ↑			
Generator	**Res.**	**Fixed**	**FARE**	**Oracle**
SD 2.1	12.7	10.2	**13.8** (+3.6)	*20.4*
SDXL	24.6	25.8	**29.4** (+3.6)	*38.1*
FLUX.1-dev	35.7	37.4	**42.0** (+4.6)	*52.5*

Table 3: **Early trajectory pruning.** Rec. denotes recovery, Ret. recovery retention, and Time average post-failure latency on an NVIDIA A100.

Method	**Rec. (%)**	**Ret. (%)**	**Time (s)**
No Pruning	74.5	100.0	6.4
Generic Early Score	71.2	95.6	5.2
Shuffled Failure Ref.	71.5	96.0	5.2
Failure Ref.	72.5	97.3	**5.1**
Failure Ref. + Cons.	**73.5**	**98.7**	5.2

across evaluation settings. FARE w/o pruning achieves 13.8%, 25.4%, and 76.2% recovery on GenEval2, TIFA, and CompBench, respectively, improving over the strongest fixed policy by 1.1, 0.2, and 0.9 percentage points. Compared with feedback-based and inference-time scaling baselines, FARE obtains the highest recovery on GenEval2 and CompBench and remains within 0.1 point of the best result on TIFA. These results support request-specific allocation rather than committing the entire recovery budget to a single intervention.

Efficiency. Although all methods share the same maximum post-failure generation budget, their realized inference costs differ. Methods that exhaust all nine SD 2.1 recovery generations require 450 denoising steps, whereas adaptive procedures may terminate before using the full budget. We therefore report both vision-language-model usage and the realized average number of denoising steps separately. FARE w/o pruning uses the full 450-step generation budget, while failure-referenced pruning reduces the realized denoising cost by terminating trajectories that remain strongly and consistently similar to the initial failed trajectory. The resulting recovery–compute trade-off is analyzed separately in Sec. 4.5.

Generalization across generators. We examine whether the benefit of failure-aware allocation persists across different T2I generators. Table 2 compares Always Resample, the train-selected fixed allocation, FARE, and a retrospective oracle on GenEval2 under the same $B = 9$ recovery budget. The oracle selects, for each request, the best feasible allocation in hindsight using the realized recovery outcomes, and therefore serves as an upper bound on allocation performance under the same generation budget. FARE improves over the train-selected fixed allocation by 3.6, 3.6, and 4.6 recovery points on SD 2.1, SDXL, and FLUX.1-dev, respectively. Absolute recovery also increases across the evaluated generators, from 13.8% with SD 2.1 to 29.4% with SDXL and 42.0% with FLUX.1-dev. The consistent improvement over the fixed allocation indicates that the benefit of request-specific recovery allocation is not specific to a single generator. Meanwhile, the remaining gap to the retrospective oracle shows that substantial instance-dependent allocation headroom remains.

4.4 Scaling Post-Failure Recovery with Inference Compute

We next ask whether failure-aware allocation remains useful as more post-failure inference compute becomes available. We increase the recovery budget from the default $B = 9$ to $B \in \{18, 27\}$ while keeping the generator and denoising schedule fixed, and compare Always Resample, Always Rewrite, and FARE in Figure 3. Recovery increases with additional compute for all three strategies. For FARE, increasing B from 9 to 27 improves recovery by 4.5–4.6 points across the three benchmarks. Most of the gain occurs by $B = 18$ (3.2–3.4 points), while the additional increase to $B = 27$ contributes only 1.2–1.4 points, indicating diminishing returns. The stronger fixed intervention varies with the benchmark and budget: resampling remains stronger on GenEval2 and TIFA, while CompBench favors rewriting at $B = 9$ but resampling at

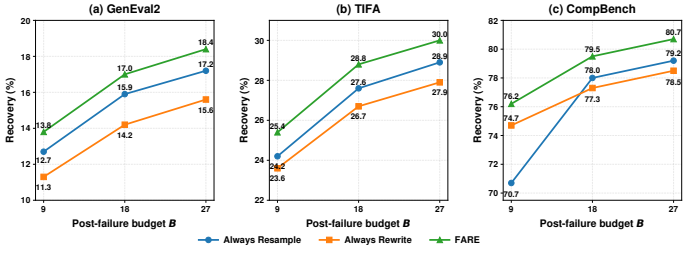


Figure 3: **Post-failure recoverability under matched budgets.**

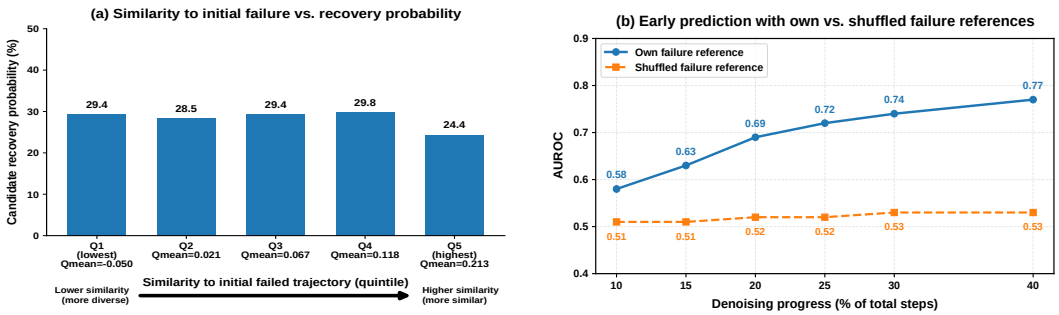


Figure 4: **Failure-referenced early trajectory signals.**

larger budgets. FARE nevertheless outperforms the stronger fixed strategy at every evaluated budget by 1.1–1.5 points. Thus, additional inference compute improves recovery but does not eliminate the allocation problem: how the budget is distributed across seed- and prompt-space exploration remains important.

4.5 Failure-Referenced Trajectory Pruning for Compute-Efficient Recovery

We first examine whether the initial failed trajectory provides an informative reference for identifying unlikely recovery candidates. Figure 4(a) groups recovery trajectories by their similarity to the corresponding initial failure. Recovery remains relatively stable across the first four similarity quintiles (28.5–29.8%), but drops to 24.4% in the highest-similarity quintile. This suggests that candidates that most strongly reproduce the observed failure trajectory are less likely to recover the original request. Figure 4(b) further tests whether this signal is request-specific by replacing each candidate's matched failed trajectory with a failure trajectory from another request. Prediction degrades when the matched reference is shuffled, showing that the observed initial failure contributes information beyond generic early-trajectory statistics. Together with the trend in Figure 4(a), this supports using the initial failure as a request-specific reference rather than relying only on a generic early-quality signal.

We then evaluate whether this signal supports compute-efficient early pruning. Table 3 compares pruning strategies on the same $B = 9$ recovery pool. Without pruning, recovery is 74.5% at 6.4 s per request. Generic early scoring reduces latency to 5.2 s with 95.6% retention, while a shuffled failure reference gives a similar 96.0%. Using the matched initial failure improves retention to 97.3% and recovery to 72.5% at 5.1 s. Adding temporal consistency further raises retention to 98.7% and recovery to 73.5% at 5.2 s. These results show that failure-referenced pruning improves the recovery–latency trade-off rather than simply pruning more aggressively. At the same 5.2 s latency as generic early scoring, Failure Ref. + Cons. retains more recoverable requests while reducing latency by 18.8% relative to full denoising. The matched failure therefore provides a more selective pruning signal, while temporal consistency protects promising candidates from premature termination. Together with failure-aware allocation, FARE uses the initial failure to guide both where recovery compute is allocated and which trajectories continue receiving it. Additional analyses of trajectory representations, checkpoints, failure-reference ablations, and recovery–cost trade-offs are provided in Appendix D.

5 Conclusion & Limitation

We view post-failure T2I generation as a problem of how to allocate additional inference compute. Seed resampling and prompt rewriting provide complementary recovery, motivating adaptive selection rather than uniform prompt refinement. FARE uses the observed failure to guide both recovery allocation and trajectory pruning. More broadly, prompt rewriting can be treated as one intervention within a larger inference-time search space. The initial failure informs not only how to revise the prompt, but also whether rewriting is preferable to continued stochastic exploration. Our study has two limitations. First, we focus on inference-time recovery with frozen generators and do not compare against training- or fine-tuning-based methods. Second, FARE does not specify how to handle requests that remain unrecovered after the recovery budget is exhausted. Future work could study when to expand the budget, invoke stronger models, request user guidance, or terminate recovery.

AI USE STATEMENT

Generative AI tools were used to assist with drafting and polishing portions of the manuscript and to provide feedback on experimental design and the presentation and interpretation of experimental results. All methodological decisions, experiments, reported results, and AI-assisted text were independently reviewed and verified by the authors. The authors take full responsibility for the final content of this work.

REFERENCES

Chieh-Yun Chen, Min Shi, Gong Zhang, and Humphrey Shi. T2i-copilot: A training-free multi-agent text-to-image system for enhanced prompt interpretation and interactive generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 19396–19405, 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.

Rogerio Guimaraes and Pietro Perona. Inference-time scaling of diffusion models via progressive seed pruning. *arXiv preprint arXiv:2607.21591*, 2026.

Huanlei Guo, Hongxin Wei, and Bingyi Jing. Toward early quality assessment of text-to-image diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 38410–38419, June 2026.

Yushi Hu et al. Tifa: Accurate and interpretable text-to-image faithfulness evaluation with question answering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Bukun Huang, Benlei Cui, Zhizeng Ye, Xuemei Dong, Tuo Chen, Hui Xue, Dingkang Yang, Longtao Huang, Haiwen Hong, and Jingqun Tang. Diffusion probe: Generated image result prediction using cnn probes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 35926–35935, 2026.

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.

Liyao Jiang, Ruichen Chen, Chao Gao, and Di Niu. RAISE: Requirement-adaptive evolutionary refinement for training-free text-to-image alignment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22038–22048, 2026.

Amita Kamath, Kai-Wei Chang, Ranjay Krishna, Luke Zettlemoyer, Yushi Hu, and Marjan Ghazvininejad. Geneval 2: Addressing benchmark drift in text-to-image evaluation. *arXiv preprint arXiv:2512.16853*, 2025.

Joowon Kim, Ziseok Lee, Donghyeon Cho, Sanghyun Jo, Yeonsung Jung, Kyungsu Kim, and Eunho Yang. Early timestep zero-shot candidate selection for instruction-guided image editing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025.

Subin Kim, Sangwoo Mo, Mamshad Nayeem Rizve, Yiran Xu, Difan Liu, Jinwoo Shin, and Tobias Hinz. Rethinking prompt design for inference-time scaling in text-to-visual generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22090–22099, 2026.

Minghan Li, Jeremy Moebel, and Mengyu Wang. Rethinking one-step image editing through chordedit: Reproduction, simplification, and new insights, 2026. URL https://arxiv.org/abs/2606.14042.

Haoyue Liu, Xiaoyu Ma, Ye Chen, Shuguang Cui, and Xiaoying Tang. One rewrite to fix them all? type-aware repair allocation for text-to-image prompt optimization. *arXiv preprint arXiv:2607.18724*, 2026.

Liangsi Lu, Xuhang Chen, Minzhe Guo, Shichu Li, Jingchao Wang, and Yang Shi. Chordedit: One-step low-energy transport for image editing. *arXiv preprint arXiv:2602.19083*, 2026.

Nanye Ma, Shangyuan Tong, Haolin Jia, Hexiang Hu, Yu-Chuan Su, Mingda Zhang, Xuan Yang, Yandong Li, Tommi Jaakkola, Xuhui Jia, et al. Inference-time scaling for diffusion models beyond scaling denoising steps. *arXiv preprint arXiv:2501.09732*, 2025.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022.

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.

Shiyu Wu, Mingzhen Sun, Weining Wang, Yequan Wang, and Jing Liu. Visualprompter: Semantic-aware prompt optimization with visual feedback for text-to-image synthesis. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=hIwVFRLaFy.

Le Zhuo, Liangbing Zhao, Sayak Paul, Yue Liao, Renrui Zhang, Yi Xin, Peng Gao, Mohamed Elhoseiny, and Hongsheng Li. From reflection to perfection: Scaling inference-time optimization for text-to-image diffusion models via reflection tuning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 15329–15339, 2025.

A Additional Experimental Details

A.1 Benchmarks and Initial-Failure Construction

Our main experiments evaluate post-failure recovery on GenEval2 (Kamath et al., 2025), T2I-CompBench (Huang et al., 2023) (hereafter CompBench), and TIFA (Hu et al., 2023). The recoverability analysis in Sec. 4.2 additionally reports results on DrawBench-200, a stratified 600-prompt subset of TIFA (TIFA-600), and 2,400 examples from CompBench. The generator-generalization experiment evaluates GenEval2 with SD2.1, SDXL, and FLUX.1-dev.

For every benchmark–generator configuration, we first generate exactly one image from the original prompt using a fixed initial seed $s_0 = 0$. Let

$$I_0^{(m)}(p) = G_m(p, s_0) \tag{1}$$

denote the initial generation produced by generator m. The corresponding semantic verifier is then applied to the original prompt and the generated image. We define the generator-specific initial-failure set as

$$\mathcal{F}_m = \left\{ p \in \mathcal{D} \mid V(p, I_0^{(m)}(p)) = 0 \right\}. \tag{2}$$

Only requests in $\mathcal{F}_m$ are passed to the recovery stage. Within each benchmark–generator configuration, all compared recovery methods operate on the same initial-failure set and share the same seed-0 initial image.

For GenEval2, we evaluate the full set of 800 prompts. Using the semantic verification protocol described in Appendix A.3, the SD2.1 configuration yields 761 initial failures, while FLUX.1-dev yields 655. Generator-specific failure sets are constructed independently, since the same request may succeed for one generator and fail for another. The shared initial generation is used only to define the failure set and provide the initial failure evidence; it is excluded from the post-failure recovery budget.

A.2 Generation Settings and Recovery Budget

Unless otherwise specified, the main experiments use Stable Diffusion 2.1-base (`stabilityai/stable-diffusion-2-1-base`) implemented with the HuggingFace Diffusers `StableDiffusionPipeline`. Images are generated at 512×512 resolution with a DDIM scheduler, classifier-free guidance scale 7.5, and $T = 50$ denoising steps. The same configuration is used for original-prompt resampling and rewritten-prompt generation. For FLUX.1-dev, images are generated at 1024×1024 resolution using 28 inference steps and guidance scale 3.5. Within each generator setting, all compared recovery policies use the same generation configuration.

The initial image always uses the fixed seed $s_0 = 0$ and is never reused as a recovery candidate. Additional recovery candidates use deterministic integer seeds beginning from $s = 1$, with a separate random generator initialized for each image. For a branch receiving b_j candidate generations, the corresponding recovery seeds are the prefix $1, \ldots, b_j$. Different recovery branches may therefore use the same numeric seed, but they correspond to different conditioning prompts and independent generation trajectories.

The post-failure budget B specifies the maximum image-generation compute available after the initial failure. One full-generation equivalent (FGE) corresponds to one complete denoising trajectory under the corresponding generator. The default experiments use $B = 9$ FGEs. For SD2.1, one FGE contains $T = 50$ denoising steps, so completing all nine recovery trajectories corresponds to

$$9 \times 50 = 450 \tag{3}$$

post-failure denoising steps. The inference-time scaling experiments additionally evaluate $B \in \{18, 27\}$ while keeping the generator and denoising schedule fixed. The budget is a maximum rather than a requirement: when a trajectory is terminated before completion, only the denoising steps actually executed are counted toward the realized generation cost.

A.3 Semantic Verification and Failure Diagnosis

All generated recovery candidates are evaluated against the original user prompt, including candidates generated from rewritten prompts. A rewritten prompt therefore changes only the conditioning used for image generation and never replaces the downstream evaluation target. Within each benchmark, all compared methods use the same semantic evaluation protocol.

GenEval2 decomposes each prompt into a collection of atomic visual question-answering constraints with semantic skill annotations. We use Soft-TIFA with Qwen3-VL-8B-Instruct to evaluate each generated image against the original VQA annotations. For atomic constraint i, the verifier produces a soft score $s_i \in [0, 1]$ corresponding to the probability mass assigned to the gold answer. Equivalent numeric answer forms are aggregated for counting questions, and equivalent affirmative forms are aggregated for binary questions. Given M atomic constraints, we compute

$$S_{\mathrm{AM}} = \frac{1}{M} \sum_{i=1}^{M} s_i, \qquad S_{\mathrm{GM}} = \left(\prod_{i=1}^{M} s_i \right)^{1/M}. \tag{4}$$

The geometric-mean score is used for prompt-level verification. A generation is considered successful when

$$S_{\mathrm{GM}} \geq \tau_{\mathrm{succ}}, \qquad \tau_{\mathrm{succ}} = 0.5. \tag{5}$$

For TIFA and CompBench, recovery is evaluated using the corresponding benchmark semantic annotations under the same original-prompt evaluation principle.

For a failed generation, an atomic constraint is marked as violated when its semantic score is below the success threshold. The violated atoms and their associated semantic categories form the structured failure evidence e_0. We map the semantic annotations to five failure facets,

$$\{\text{object}, \text{attribute}, \text{count}, \text{relation}, \text{action}\}, \tag{6}$$

where position is mapped to relation and verb is mapped to action. The complete initial-failure evidence is

$$E_0 = (p, I_0, e_0). \tag{7}$$

This evidence is computed before any recovery generation and is shared by the failure-aware allocation and trajectory-pruning stages. Details of the failure-conditioned rewrite operators are provided in Appendix B, while the allocation features constructed from E_0 are described in Appendix C.

A.4 Evaluation Metrics and Compute Accounting

Our primary metric is request-level recovery, evaluated only on the shared initial-failure subset. Let $\mathcal{E}_m \subseteq \mathcal{F}_m$ denote the evaluated initial-failure requests for generator m, and let $\mathcal{C}_{\pi,m}(p)$ denote the completed recovery candidates produced by policy π for request p. For a candidate image x, define

$$Y(x,p) = \mathbb{1}[V(p,x) = 1], \tag{8}$$

where verification is always performed against the original prompt. The request-level recovery rate is

$$\mathrm{Recovery}(\pi, m) = \frac{1}{|\mathcal{E}_m|} \sum_{p \in \mathcal{E}_m} \mathbb{1}\left[\exists x \in \mathcal{C}_{\pi,m}(p) : Y(x,p) = 1\right]. \tag{9}$$

A request is therefore counted as recovered if at least one completed candidate satisfies the semantic verifier. Unavailable rewrite branches or infeasible allocations are not assigned negative recovery labels.

We additionally retain the continuous semantic scores produced by the benchmark evaluator as diagnostic measures of image–prompt alignment. For oracle comparisons, the retrospective oracle selects the best feasible allocation for each request using the realized candidate outcomes under the same post-failure generation budget. The oracle is used only as an upper bound and is not available at inference time.

For trajectory-pruning experiments, we also report recovery retention. Let $\mathcal{R}_{\mathrm{full}}$ denote the set of requests recoverable by the complete unpruned candidate pool and let $\mathcal{R}_\pi$ denote the set recovered after applying pruning policy π. Retention is defined as

$$\mathrm{Retention}(\pi) = \frac{|\mathcal{R}_\pi \cap \mathcal{R}_{\mathrm{full}}|}{|\mathcal{R}_{\mathrm{full}}|}. \tag{10}$$

Both sets are computed using the same evaluated requests and underlying candidate pool.

Generation cost is measured by the number of denoising steps actually executed after the initial failure. Let T_m denote the number of denoising steps in one complete generation for generator m, and let ℓ_i be the number of steps executed by recovery candidate i. For request p, the realized denoising cost is

$$\mathrm{DSteps}(p) = \sum_i \ell_i, \tag{11}$$

and the corresponding cost in full-generation equivalents is

$$C_{\mathrm{FGE}}(p) = \frac{\sum_i \ell_i}{T_m}. \tag{12}$$

A completed trajectory therefore contributes one FGE, whereas a candidate terminated after $\ell_i < T_m$ steps contributes only ℓ_i / T_m. For SD2.1, $T_m = 50$. The conditional and unconditional branches used for classifier-free guidance are not counted separately. The seed-0 initial generation is excluded from all post-failure DSteps and FGE measurements.

Vision-language-model usage is reported separately from diffusion generation. VLM Calls in the main comparison count image-level VLM invocations used internally by the recovery procedure and exclude the shared final semantic evaluator. A single image-level judge job may evaluate all VQA questions associated with that image. LLM-based text rewriting is likewise not converted into diffusion full-generation equivalents. Thus, DSteps/FGE and VLM Calls measure distinct components of inference-time computation.

A.5 Matched-Budget Experimental Protocol

All compared methods are evaluated under the same post-failure protocol. For a given benchmark–generator configuration, they start from the same seed-0 initial image, operate on the same initial-failure subset, and use the same underlying image generator and generation settings. The initial

image is excluded from the recovery budget. Whenever an additional stochastic generation is required, recovery seeds follow the deterministic protocol described in Appendix A.2.

The maximum recovery budget is shared across the complete recovery procedure rather than assigned independently to individual refinement rounds. For an iterative recovery procedure that generates n_r candidates at round r, the total number of generated candidates satisfies

$$\sum_r n_r \leq B. \tag{13}$$

Candidates generated before and after prompt modification, semantic feedback, or other intermediate recovery operations therefore consume the same common image-generation budget. For methods that combine multiple recovery mechanisms, all generated images jointly count toward this budget. When a baseline includes cross-model generator selection, that component is disabled in the common-generator comparison so that image-generation capacity is held fixed across methods.

All completed images are evaluated with the same benchmark-specific semantic verifier against the original user prompt. Intermediate prompt modifications, visual feedback, or candidate selection operations do not alter the evaluation target. Generation cost is determined only by the denoising steps actually executed. Methods that complete the full $B = 9$ SD2.1 budget therefore incur 450 post-failure denoising steps, while methods with early stopping or trajectory termination may realize a smaller cost. Internal VLM usage is reported separately, and neither semantic feedback calls nor text-rewriting calls increase the available diffusion-generation budget.

Finally, all candidate outcomes and derived supervision associated with the same original request are kept in the same data split. This includes recovery candidates, rewrite branches, allocation labels, and trajectory features. Allocation models are trained using training requests, and decision thresholds and pruning operating points are selected using training or validation data only. These choices are fixed before evaluation on the held-out test requests.

B Failure-Conditioned Rewrite Operators

B.1 Overview of Rewrite Operators

Motivation for multiple semantic interventions. A single rewrite strategy is unlikely to address the diverse semantic failures captured by Soft-TIFA. GenEval2 failures span multiple facets, including counting, attribute binding, spatial relations, and actions. We therefore employ three complementary, semantics-preserving rewrite operators conditioned on the original prompt and the diagnosed failed atoms: **O1: Local Reformulation**, which applies localized lexical or syntactic modifications; **O2: Failure Reinforcement**, which explicitly restates the violated constraints; and **O3: Structural Reorganization**, which reorganizes the prompt structure while preserving the original semantic requirements. Each operator is equipped with an availability gate, and only gate-passing rewrites are included in the recovery candidate pool.

Rewrite as internal search. Prompt rewriting does not alter the evaluation target. Instead, it expands an internal search space over semantically equivalent prompt variants under a fixed post-failure generation budget B. Together with resampling the original prompt, the allocation stage determines how many of the B generation slots are assigned to the original prompt and to the O1/O2/O3 rewrite branches. Rewriting therefore serves as a structured hypothesis-search mechanism within the recovery budget, rather than modifying the downstream success criterion.

Evaluation against the original prompt. Every rewritten prompt is used only as an alternative conditioning signal for image generation. The resulting image is always evaluated with Soft-TIFA using the *original prompt's* VQA list, rather than questions derived from the rewritten prompt. A candidate is considered successful if

$$S_{\mathrm{GM}} \geq \tau_{\mathrm{succ}}, \tag{14}$$

where S_{GM} denotes the Soft-TIFA geometric-mean score and $\tau_{\mathrm{succ}} = 0.5$. This design ensures that recovery remains faithful to the user's original compositional intent: a rewrite may provide a more effective conditioning representation, but success is determined solely by whether the generated image satisfies the original atomic semantic constraints.

B.2 LOCAL REPAIR (o_1)

Definition. Local Repair (o_1) performs a localized lexical or syntactic reformulation of prompt spans associated with the diagnosed failed constraints. Given the original prompt p and the failed-atom set $\mathcal{F}$, it produces a rewritten prompt $p^{(1)}$ that aims to express the same semantic requirements in a form that is easier for the T2I generator to realize. Unlike semantic reinforcement or structural rewriting, o_1 modifies only failure-relevant phrasing and leaves the overall prompt structure unchanged.

Applicable failures. Local Repair is applicable when the required semantic content is already present in the prompt but may be difficult to realize because of its wording or local syntactic form. Typical cases include failures involving counts, attributes, spatial relations, colors or textures, and actions or roles. Failure facets are provided to the rewriter as diagnostic context, but do not directly determine whether o_1 is activated.

Rewrite constraints. The operator may substitute failure-related expressions with semantically equivalent alternatives, locally reorder phrases, or clarify ambiguous references. It must preserve all original objects, attributes, counts, relations, actions, and stylistic requirements. It is not allowed to repeat constraints for emphasis, globally restructure the prompt, introduce unsupported content, simplify rare actions into different actions, or return the original prompt unchanged.

Examples.

- $p =$ *"seven green croissants"*, with failed count and color constraints,
$$p^{(1)} = \textit{"seven green-colored croissants"}.$$
- $p =$ *"four rabbits and a sheep"*, with a failed counting constraint,
$$p^{(1)} = \textit{"four rabbits and one sheep"}.$$

B.3 SEMANTIC REINFORCEMENT (o_2)

Definition. Semantic Reinforcement (o_2) retains the original prompt as the semantic backbone while explicitly restating the diagnosed failed constraints in a short, logically equivalent form. The resulting prompt $p^{(2)}$ increases the salience of constraints that may be weakly expressed or insufficiently bound, without changing the intended scene content.

Failure-conditioned reinforcement. The form of reinforcement is determined by the failed atoms in $\mathcal{F}$:

- **Count:** restate the required cardinality explicitly, e.g., "exactly four rabbits" or "one sheep is present."
- **Attribute:** explicitly re-establish the binding between an attribute and its intended entity, including color, material, or texture.
- **Relation:** restate a spatial or semantic relation using an equivalent formulation, e.g., "A is left of B" together with "B is right of A," or explicitly clarify the corresponding subject–object roles.

Rewrite constraints. The original description remains the basis of the rewrite, and only failed constraints may be reinforced. The operator may append or integrate short semantically equivalent clarifications, but must not introduce new objects, attributes, camera specifications, or layout constraints. It must not delete existing requirements, globally reorganize the prompt, or return a rewrite without any meaningful reinforcement.

Examples.

- $p =$ *"two metal toys"*, with a failed counting constraint,
$$p^{(2)} = \textit{"two metal toys (There are exactly two toys in the image.)"}.$$

- $p =$ *"four rabbits and a sheep"*,

  $p^{(2)} =$ *"four rabbits and a sheep (exactly four rabbits and one sheep are present)"*.

B.4 STRUCTURAL REWRITE (o_3)

Definition. Structural Rewrite (o_3) changes how the original semantic constraints are organized in text while preserving their meaning. Given p and $\mathcal{F}$, it produces $p^{(3)}$ by decomposing a densely packed natural-language description into a clearer compositional structure, typically separating *entity–attribute* clauses from *relation* clauses.

This operator targets prompts in which multiple entities, attributes, counts, or relations are expressed within a single compact sentence and textual packing may make the intended composition difficult for the generator to realize.

Structural reformulation. Entities and their attributes are expressed as explicit local clauses, while spatial, relational, or role constraints are stated separately. Counts remain attached to the corresponding entities. The purpose is to modify the textual organization of the constraints, rather than their semantic content.

Meaning preservation. Structural Rewrite must preserve all requirements of the original prompt, including objects, attributes, counts, relations, actions, and style. It must not introduce unsupported assumptions about location, visibility, camera configuration, or scene composition; replace rare actions with different actions; remove constraints; or return the original prompt unchanged.

Examples.

- $p =$ *"seven green croissants"*,

  $p^{(3)} =$ *"seven croissants; the croissants are green"*.

- $p =$ *"four rabbits and a sheep"*,

  $p^{(3)} =$ *"four rabbits; one sheep"*.

B.5 OPERATOR AVAILABILITY

Semantic-preservation gate. For each failed request i, we attempt to construct one rewrite from each operator $o_k \in \{o_1, o_2, o_3\}$. A generated rewrite $p_i^{(k)}$ enters the recovery candidate pool only if it passes a two-stage semantic-preservation gate.

The first stage applies lightweight heuristic checks: the rewrite must be non-empty, differ from the normalized original prompt, and satisfy a minimum length requirement relative to the original prompt. The second stage uses an LLM-based semantic checker to verify that the rewrite is a faithful reformulation of p. A rewrite is rejected if it is effectively identical to the original, drops an important constraint, introduces unsupported content, or contradicts the original prompt.

If a rewrite fails the gate, the operator may be retried once with feedback indicating the reason for rejection. If no valid rewrite is obtained, that operator is marked unavailable for the corresponding request.

Failure types and operator availability. Failure facets are not used as hard routing rules. Instead, diagnostic categories such as COUNT, SPATIAL, ATTRIBUTE_BINDING, COLOR_OR_TEXTURE, and ACTION_OR_ROLE are provided as contextual information to all three operators. Consequently, operator availability is determined by whether a semantics-preserving rewrite can be constructed for the particular request, rather than by a fixed mapping from failure type to operator.

Empirically, the operators provide different inductive biases: o_1 favors localized reformulation, o_2 makes failed constraints more explicit, and o_3 reorganizes densely composed prompts. However, these tendencies do not constitute hard activation rules.

Feasible operator set. Let

$$\mathrm{avail}_i(o_k) \in \{0, 1\}$$

denote the gate outcome for request i and operator o_k. The set of feasible rewrite operators is then

$$\mathcal{O}_i = \{o_k \mid \mathrm{avail}_i(o_k) = 1\}. \tag{15}$$

The downstream allocation stage may assign budget only to operators in $\mathcal{O}_i$. Unavailable operators receive zero allocation and are excluded from the candidate pool.

B.6 Rewrite Prompt Templates

Shared input representation. All three rewrite operators receive the same structured information: the original user prompt, the semantic decomposition of that prompt, the constraints violated by the initial generation, and their mapped failure types. The operator-specific instruction then determines how this information is used to construct the rewrite.

The shared input format is:

```
ORIGINAL PROMPT:
{prompt}

SEMANTIC DRAFT:
{1. facet | tuple/span | question
 ...}

FAILED CONSTRAINTS:
{1. facet | tuple/span | question
 ...}

FAILURE TYPE:
{COUNT, SPATIAL, ...}
```

Operator-specific instructions. o_1 — *Local Repair:*

```
Rewrite the text-to-image prompt using LOCAL REFORMULATION only.

Goal:
Make the failed constraints easier to realize without changing
the scene semantics.

Allowed:
- Replace failure-related expressions with equivalent wording.
- Apply local syntactic reformulation.
- Clarify ambiguous pronouns or references.
- Preserve every original object, attribute, count, relation,
  action, and style requirement.

Forbidden:
- Do not repeat constraints for emphasis.
- Do not reorganize the entire prompt.
- Do not add unsupported objects, attributes, camera settings,
  or layout constraints.
- Do not replace rare actions with different actions.
- Do not return the original prompt unchanged.

Return only the rewritten prompt as a single line.
```

o_2 — *Semantic Reinforcement:*

```
Rewrite the text-to-image prompt using SEMANTIC REINFORCEMENT.
```

```
Goal:
Retain the original description while making the failed
constraints more explicit.

Allowed:
- Keep the original wording as the semantic backbone.
- Briefly restate failed constraints in an equivalent form.
- Clarify attribute--entity bindings.
- Restate spatial relations using equivalent formulations.

Forbidden:
- Do not add unsupported objects, attributes, camera settings,
  or layout constraints.
- Do not remove any original constraint.
- Do not globally restructure the prompt.
- Do not return the prompt without meaningful reinforcement.

Return only the rewritten prompt as a single line.
```

o_3 — *Structural Rewrite:*

```
Rewrite the text-to-image prompt using STRUCTURAL REORGANIZATION.

Goal:
Reorganize the prompt into clear entity--attribute clauses
followed by relation or action clauses.

Allowed:
- Separate entities and their attributes into explicit clauses.
- Express spatial, relational, or role constraints separately.
- Preserve all original semantic requirements.

Forbidden:
- Do not introduce unsupported location or visibility assumptions.
- Do not replace rare actions with different actions.
- Do not remove any original constraint.
- Do not return the original prompt unchanged.

Return only the rewritten prompt as a single line.
```

Output and validation. Each operator returns exactly one single-line rewritten prompt. The semantic-preservation checker then outputs one of

`PASS: <reason>` or `FAIL: <reason>`.

Only rewrites receiving `PASS` are retained as feasible recovery hypotheses.

Examples. Table 4 illustrates representative operator outputs and availability outcomes.

C Additional Details of Failure-Aware Budget Allocation

This section provides implementation details for the failure-aware allocation module introduced in Sec. 3.2. Given the initial failure evidence $E_0 = (p, I_0, e_0)$, the allocation module predicts the recovery utility of feasible portfolios over original-prompt resampling and three failure-conditioned rewrite branches. All allocation decisions are made before recovery generation, and all candidate outcomes used for supervision are evaluated against the original user prompt.

Table 4: **Examples of failure-conditioned rewrite operators.** A check mark indicates that the generated rewrite passes the semantic-preservation gate.

Original prompt	Failure	Rewrite
seven green croissants	Count, Attr.	o_1: *seven green-colored croissants* ✓ o_2: rejected by gate o_3: *seven croissants; the croissants are green* ✓
two metal toys	Count	o_1: rejected by gate (dropped *metal*) o_2: *two metal toys (There are exactly two toys in the image.)* ✓ o_3: rejected by gate
four rabbits and a sheep	Count, Attr.	o_1: *four rabbits and one sheep* ✓ o_2: *four rabbits and a sheep (exactly four rabbits and one sheep are present)* ✓ o_3: *four rabbits; one sheep* ✓

C.1 Feasible Allocation Space

FARE considers four recovery branches: original-prompt resampling and the three failure-conditioned rewrite operators o_1, o_2, and o_3, corresponding to Local Reformulation, Failure Reinforcement, and Structural Reorganization. An allocation is a non-negative integer tuple

$$a = (b_0, b_1, b_2, b_3), \qquad \sum_{j=0}^{3} b_j = B, \qquad b_j \in \mathbb{Z}_{\geq 0}, \tag{16}$$

where b_0 denotes the number of original-prompt resamples and b_1, b_2, b_3 denote the numbers of samples assigned to the three rewrite branches.

Without operator-availability constraints, the number of possible allocations is

$$|\mathcal{A}(B)| = \binom{B+3}{3}. \tag{17}$$

For the default budget $B = 9$, this gives 220 allocations. The space includes always-resample,

$$a_{\mathrm{AR}} = (9, 0, 0, 0), \tag{18}$$

as well as allocations that divide the budget across multiple original and rewritten prompts.

Rewrite availability is determined independently for each request using the semantic-preservation gates described in Appendix B. Let $u_{ij} \in \{0, 1\}$ denote whether rewrite operator o_j is available for request i. The request-specific feasible allocation set is

$$\mathcal{A}_i(B) = \{a \in \mathcal{A}(B) : b_j = 0 \text{ if } u_{ij} = 0,\ j \in \{1, 2, 3\}\}. \tag{19}$$

If r_i rewrite operators are available, the number of feasible allocations is

$$|\mathcal{A}_i(B)| = \binom{B+r_i}{r_i}. \tag{20}$$

At $B = 9$, this corresponds to 220, 55, 10, or 1 feasible allocations when three, two, one, or zero rewrite operators are available. Always-resample remains feasible for every request.

C.2 Counterfactual Recovery Supervision

For every initial-failure request, we construct candidate pools for the original prompt and each available rewrite branch. Under the default $B = 9$ setting, each branch contains recovery seeds $s = 1, \ldots, 9$. Candidate images are generated independently but are evaluated using the semantic verifier associated with the original user prompt.

Table 5: **Feature configurations for allocation prediction.** F2 is used for the main FARE results.

Mode	Composition	Dim.
F0	Initial failure evidence + allocation	55
F1	F0 + rewrite-side features	95
F2	F1 + allocation–rewrite interactions	105
F2+Text	F2 + original-prompt embedding	617

Let x_{ijs} denote the candidate produced for request i, branch j, and recovery seed s, and let

$$z_{ijs} = \mathbb{1}\left[S(p_i, x_{ijs}) \geq \tau_{\text{succ}}\right] \tag{21}$$

denote whether that candidate satisfies the original prompt. For GenEval2, S is the Soft-TIFA geometric-mean score and $\tau_{\text{succ}} = 0.5$.

An allocation $a = (b_0, b_1, b_2, b_3)$ selects the corresponding prefix from each branch:

$$\mathcal{C}_i(a) = \{x_{ijs} : j \in \{0,1,2,3\},\ 1 \leq s \leq b_j\}\,. \tag{22}$$

Its counterfactual recovery label is

$$Y_i(a) = \mathbb{1}\left[\sum_{j=0}^{3}\sum_{s=1}^{b_j} z_{ijs} > 0\right]. \tag{23}$$

Thus, $Y_i(a) = 1$ if at least one candidate contained in allocation a succeeds under the original prompt.

Candidate ordering is fixed by recovery seed and is not selected using observed downstream outcomes. Candidate outcomes shared by multiple allocations are cached and reused when constructing counterfactual labels. Allocations requiring unavailable rewrite branches or missing candidate generations are marked infeasible and do not enter predictor training.

All allocations associated with the same original request remain in the same data split. Candidate semantic scores and recovery labels are used only to construct supervision and evaluation targets; they are never included as predictor inputs.

C.3 Allocation Predictor Features

The predictor represents each request–allocation pair using initial-failure evidence, allocation structure, and rewrite-side information. We construct cumulative feature configurations from F0 to F2. The feature definitions are summarized in Table 5.

The allocation block contains the four branch budgets, total budget, number of active branches, branch-budget shares, total rewrite share, maximum branch share, and allocation entropy. For $s_j = b_j/B$, allocation entropy is

$$H(a) = -\sum_{j=0}^{3} s_j \log s_j, \tag{24}$$

with $0 \log 0 = 0$.

The initial-failure block describes the semantic state of I_0 before recovery. It includes the number and fraction of violated atomic constraints, initial semantic score, and per-facet counts and indicators describing object, attribute, count, relation, and action failures. These features are computed exclusively from E_0.

Rewrite-side features describe the available recovery hypotheses before candidate generation. For each operator, they include availability, the failure facets targeted by the rewrite, rewrite length and length change, edit magnitude, and semantic-preservation-gate information. The interaction block additionally combines the allocation with these rewrite descriptors, including the amount of budget assigned to available operators and to rewrites targeting each diagnosed failure facet.

The optional text variant appends a 512-dimensional normalized text embedding of the original prompt. The main experiments use F2 so that the allocation decision is based on structured failure and rewrite information without introducing an additional prompt-embedding model.

Table 6: **Failure-aware allocation at the main $B = 9$ operating point.** Fixed denotes the train-selected fixed allocation. Δ is the recovery gain of FARE over Fixed.

Benchmark	Generator	Fixed	FARE	Δ
GenEval2	SD2.1	10.2	13.8	+3.6
TIFA	SD2.1	25.2	25.4	+0.2
CompBench	SD2.1	75.3	76.2	+0.9
GenEval2	SDXL	25.8	29.4	+3.6
GenEval2	FLUX.1-dev	37.4	42.0	+4.6

C.4 Predictor Training and Main Operating Point

We implement q_θ using a histogram-based gradient-boosted decision-tree classifier. Given the feature vector $\phi(i, a)$, the predictor outputs

$$q_\theta(i, a) = P\left(Y_i(a) = 1 \mid \phi(i, a)\right). \tag{25}$$

The classifier is trained with binary log loss over complete and feasible request–allocation pairs:

$$\mathcal{L}(\theta) = -\frac{1}{|\mathcal{T}|} \sum_{(i,a)\in\mathcal{T}} \left[Y_i(a) \log q_\theta(i, a) + (1 - Y_i(a)) \log(1 - q_\theta(i, a))\right]. \tag{26}$$

We use maximum tree depth 3, learning rate 0.08, 150 boosting iterations, minimum leaf size 8, and ℓ_2 regularization 1.0. No candidate outcome or final recovery score is used as an input feature. Splitting is performed at the request level so that all allocations, rewrites, and candidate outcomes associated with the same request remain in the same partition. Training requests are used to fit the predictor, validation requests are used to fix the operating policy, and the resulting configuration is held fixed during test evaluation.

The final experiments use the F2 feature configuration together with the conservative allocation rule in Appendix C.5. The same predictor definition and decision rule are used across the main benchmarks and generator-generalization experiments. Table 6 reports the corresponding $B = 9$ operating points used in the main paper.

C.5 Conservative Allocation Policy

For a failed request i, the predictor first identifies the highest-scoring feasible allocation

$$\tilde{a}_i = \arg\max_{a\in\mathcal{A}_i(B)} q_\theta(i, a). \tag{27}$$

Let

$$a_{\mathrm{AR}} = (B, 0, 0, 0) \tag{28}$$

denote always-resample. We compare the predicted utility of the best adaptive allocation with the resampling default:

$$\Delta_i = q_\theta(i, \tilde{a}_i) - q_\theta(i, a_{\mathrm{AR}}). \tag{29}$$

The final allocation is

$$a_i^\star = \begin{cases} \tilde{a}_i, & \Delta_i \geq \delta, \\ a_{\mathrm{AR}}, & \Delta_i < \delta. \end{cases} \tag{30}$$

We use

$$\delta = 0.02 \tag{31}$$

throughout the main experiments.

The margin prevents small predicted differences from automatically replacing a strong resampling baseline. Once $a_i^\star$ is selected, the predictor is no longer queried: the corresponding recovery candidates are launched in parallel, and their realized denoising cost may subsequently be reduced by the trajectory-pruning module.

Table 7: **Post-failure recovery across inference budgets.** All methods use the same generator and denoising schedule within each benchmark.

Benchmark	Method	$B = 9$	$B = 18$	$B = 27$
GenEval2	Always Resample	12.7	15.9	17.2
	Always Rewrite	11.3	14.2	15.6
	FARE	13.8	17.0	18.4
TIFA	Always Resample	24.2	27.6	28.9
	Always Rewrite	23.6	26.7	27.9
	FARE	25.4	28.8	30.0
CompBench	Always Resample	70.7	78.0	79.2
	Always Rewrite	74.7	77.3	78.5
	FARE	76.2	79.5	80.7

C.6 Allocation at Larger Recovery Budgets

The inference-time scaling experiments extend the same allocation formulation to $B \in \{18, 27\}$. The unrestricted allocation spaces contain

$$|\mathcal{A}(18)| = \binom{21}{3} = 1330 \tag{32}$$

and

$$|\mathcal{A}(27)| = \binom{30}{3} = 4060 \tag{33}$$

possible allocations before rewrite availability is applied.

For each evaluated budget, the branch cache is extended to contain the corresponding recovery-seed prefix $1, \ldots, B$. Counterfactual labels are constructed using the same definition as in Appendix C.2. We retain the same F2 representation and predictor architecture, while training the allocation predictor for the corresponding recovery budget rather than extrapolating a predictor trained only at $B = 9$. The conservative margin remains fixed at $\delta = 0.02$.

Table 7 reports the recovery values used in Sec. 4.4. The scaling experiment uses its fixed matched-budget evaluation pool at every value of B; consequently, results should be compared within this scaling protocol rather than across evaluation subsets.

C.7 Allocation Behavior by Initial Failure Facet

We additionally report the relative recovery behavior of resampling and rewriting across semantic facets of the initial failure. Failure facets are extracted from E_0 and may overlap within the same request. Table 8 reports the corresponding $B = 9$ results on GenEval2.

The facet counts are overlapping because one initial failure may violate multiple semantic requirements. The allocation predictor therefore uses the complete structured failure representation rather than assigning each request to a single failure category.

D Additional Details of Failure-Referenced Trajectory Pruning

This section provides implementation and evaluation details for the failure-referenced trajectory-pruning module introduced in Sec. 3.3. The initial seed-0 failed trajectory is used as a request-specific reference, and each recovery trajectory is compared with this reference during early denoising.

Unless otherwise specified, the pruning experiments use CompBench with SD2.1, DDIM sampling with $T = 50$ denoising steps, classifier-free guidance (CFG) scale 7.5, and 512×512 resolution. The seed-0 trajectory is used only as the failure reference, while recovery candidates are generated using seeds $s_1, \ldots, s_9$. All training, validation, and test partitions are constructed at the request

Table 8: **Recovery by initial failure facet.** Facets are determined from E_0 and are not mutually exclusive. $\Delta = \text{Rewrite} - \text{Resample}$.

Facet	N	Resample	Rewrite	Δ
SD2.1				
Count	739	12.2	10.6	-1.6
Object	567	9.0	6.9	-2.1
Attribute	526	5.5	5.7	+0.2
Relation	450	7.1	4.2	-2.9
Action	80	6.3	3.8	-2.5
FLUX.1-dev				
Count	581	33.9	34.4	+0.5
Relation	332	22.6	27.1	+4.5
Attribute	311	23.8	27.0	+3.2
Object	181	26.5	23.2	-3.3
Action	62	12.9	14.5	+1.6

Table 9: **Operational checkpoints for trajectory pruning.** Progress denotes the fraction of the 50-step DDIM trajectory already executed.

Step k	Progress	Label	Scheduler timestep
5	10%	$t_{0.90}$	901
8	16%	$t_{0.85}$	841
10	20%	$t_{0.80}$	801
12	24%	$t_{0.75}$	761

level. Candidate success labels are obtained from the cached downstream semantic evaluation, so trajectory-feature extraction introduces no additional final-image VLM evaluations.

The diagnostic experiments below use fixed candidate pools to isolate the trajectory signal. In the complete FARE pipeline, the same pruning rule is applied to the recovery candidates selected by the allocation module.

D.1 Early Trajectory Features and Checkpoints

We use four operational checkpoints during early denoising,

$$\mathcal{K}_{\text{op}} = \{5, 8, 10, 12\}, \tag{34}$$

corresponding to approximately 10%, 16%, 20%, and 24% of the $T = 50$ DDIM trajectory. These checkpoints cover the high-noise regime used by the operational pruning rule and approximately correspond to $t \in [0.75, 0.9]$ in the notation of Sec. 3.3.

At each checkpoint, we extract two complementary failure-reference signals. The first compares the global CFG direction. For latent x_k, the CFG direction is

$$g_k = \epsilon_{\text{cond}}(x_k, p) - \epsilon_{\text{uncond}}(x_k), \tag{35}$$

where p is the original prompt. Rather than comparing the full CFG tensor directly, we spatially pool each latent channel to 4×4 and compute the channel-wise ℓ_2 magnitude,

$$E_4(g_k)_{u,v} = \left(\sum_{d=1}^{4} \bar{g}_{k,d,u,v}^2 \right)^{1/2}, \tag{36}$$

where $\bar{g}_k$ denotes the spatially pooled direction. Flattening the resulting 4×4 energy map gives the 16-dimensional *energy4* representation. For recovery candidate c and its corresponding seed-0 failure c_0, the checkpoint-wise direction similarity is

$$s_k^{\text{dir}}(c) = \cos\left(E_4(g_k^c), E_4(g_k^{c_0})\right). \tag{37}$$

Table 10: **Candidate recovery by similarity to the initial failed trajectory.** Q1 contains candidates with the lowest aggregate failure-reference similarity and Q5 the highest.

Quintile	Q_{mean}	Success (%)
Q1	-0.050	29.4
Q2	0.021	28.5
Q3	0.067	29.4
Q4	0.118	29.8
Q5	0.213	24.4

The second signal focuses on semantic structures that were explicitly violated by the initial generation. Each failed atom in e_0 is mapped to its corresponding subject, object, attribute, relation, or action tokens in the original prompt. For multi-token atoms, their cross-attention maps are averaged; when multiple atoms fail, their associated token indices are combined into a failed-token set $\mathcal{T}_0$. At checkpoint k, the corresponding attention representation is

$$A^c_{4,k} = \text{Pool}_{4\times 4}\left(\frac{1}{|\mathcal{T}_0|}\sum_{j\in\mathcal{T}_0} A^c_k[:,:,j]\right), \tag{38}$$

and the failed-atom attention similarity is

$$s^{\text{attn}}_k(c) = \cos\left(A^c_{4,k}, A^{c_0}_{4,k}\right). \tag{39}$$

An 8×8 attention representation is additionally retained for diagnostic analysis, while the 4×4 representation is used by the default trajectory features.

D.2 Failure-Referenced Signal Analysis

We first examine whether similarity to the initial failed trajectory is associated with subsequent candidate success. For the analysis in Fig. 4(a), candidates are ranked by an aggregate failure-reference similarity score, denoted by Q_{mean}, and divided into five equal-sized groups. Table 10 reproduces the corresponding quintile statistics.

The candidate-level prediction target used in the diagnostic analyses is

$$Y_c = \mathbf{1}[\text{candidate } c \text{ succeeds under the original prompt}]. \tag{40}$$

where the label is obtained from the cached downstream semantic evaluation. The highest-similarity group has the lowest observed recovery rate, while recovery is comparatively stable across the remaining four groups.

We next test whether the predictive signal is specific to the observed failure trajectory. For the diagnostic experiment in Fig. 4(b), early recovery prediction is evaluated at

$$\rho \in \{10\%, 15\%, 20\%, 25\%, 30\%, 40\%\} \tag{41}$$

of the denoising process. The same predictor is evaluated using either the candidate's matched seed-0 failure reference or a failure trajectory from another request.

The diagnostic checkpoints in Table 11 are used to characterize how predictive information evolves during denoising. They are distinct from the four operational checkpoints in Table 9, which determine when the actual early-pruning decision is made.

We additionally evaluate individual trajectory representations on the operational trajectory cache. These standalone representation results are summarized in Table 12. They isolate the information contained in individual trajectory features and therefore use a different evaluation protocol from the progressive failure-conditioned predictor in Table 11; the AUROC values should not be compared directly across the two tables.

These auxiliary ablations use the four-checkpoint operational cache. They are retained to document the feature choices used by the pruning module rather than to reproduce the progressive diagnostic curve in Fig. 4(b).

Table 11: **Early recovery prediction with matched and shuffled failure references.** Values reproduce the diagnostic experiment in Fig. 4(b).

Denoising progress	Own reference	Shuffled reference
10%	0.58	0.51
15%	0.63	0.51
20%	0.69	0.52
25%	0.72	0.52
30%	0.74	0.53
40%	0.77	0.53

Table 12: **Trajectory representation ablation on the operational checkpoint cache.** Zero-shot rows use request-specific failure-reference similarity. Learned-probe rows use the held-out request-level split.

Representation / probe	AUROC ↑
Zero-shot failure-reference representations	
Raw CFG direction	0.5176
Low-frequency 8×8	0.5209
Energy $8/16/32$	0.526–0.529
Energy 4×4	0.5373
Failed-atom attention	0.5836
Failure-conditioned probe	
Direction only	0.4928
Direction $+E_0$	0.6332
Direction + attention	0.5567
Direction $+E_0$ + attention	0.6367

D.3 Conservative Failure-Referenced Pruning

For operational pruning, the checkpoint-wise direction and failed-atom attention similarities are first aggregated separately across the four early checkpoints. For the direction signal,

$$S_{\mathrm{dir}}(c) = \sum_{k \in \mathcal{K}_{\mathrm{op}}} w_k s_k^{\mathrm{dir}}(c), \tag{42}$$

and analogously

$$S_{\mathrm{attn}}(c) = \sum_{k \in \mathcal{K}_{\mathrm{op}}} w_k s_k^{\mathrm{attn}}(c), \tag{43}$$

with

$$(w_5, w_8, w_{10}, w_{12}) = (0.45, 0.25, 0.18, 0.12). \tag{44}$$

The weighting emphasizes evidence observed in the earliest portion of the trajectory.

The two aggregated signals are combined into the scalar failure-reference score $S(c)$ used by Algorithm 1. Specifically, we normalize the two similarity components using validation-set statistics and combine them before thresholding:

$$S(c) = \frac{1}{2}\left(\widehat{S}_{\mathrm{dir}}(c) + \widehat{S}_{\mathrm{attn}}(c)\right). \tag{45}$$

A high value of $S(c)$ indicates that the recovery trajectory remains similar to the request-specific failed trajectory. A direct failure-reference policy would therefore terminate a candidate when

$$S(c) > \tau_{\mathrm{sim}}. \tag{46}$$

A single high similarity value can arise from transient early noise. We therefore additionally measure temporal consistency using the cross-checkpoint variation of the combined similarity:

$$U(c) = \mathrm{Std}\left(s_5(c), s_8(c), s_{10}(c), s_{12}(c)\right), \tag{47}$$

Table 13: **Selected recovery–cost operating points.** Ret. denotes recovery retention relative to the unpruned candidate pool.

Method	Rec. (%)	Ret. (%)	Time (s)
No Pruning	74.5	100.0	6.4
Generic Early Score	71.2	95.6	5.2
Shuffled Failure Ref.	71.5	96.0	5.2
Failure Ref.	72.5	97.3	5.1
Failure Ref. + Cons.	73.5	98.7	5.2

where $s_k(c)$ denotes the combined checkpoint-level failure-reference similarity. The final pruning rule is

$$S(c) > \tau_{\text{sim}} \quad \wedge \quad U(c) < \tau_U. \tag{48}$$

Both thresholds are selected using validation requests and are fixed before test evaluation. A candidate satisfying the pruning condition is terminated after the last operational checkpoint; otherwise it continues to the full 50-step trajectory. We additionally retain at least one active recovery candidate for each request. Early-terminated trajectories are not decoded or evaluated by the final semantic judge and cannot directly contribute a successful recovery.

D.4 Recovery–Cost Trade-off

We evaluate the operational pruning policies on the same $B = 9$ candidate pool. Table 13 reports the validation-selected operating points used in Sec. 4.5. Recovery is measured at the request level, retention is measured relative to the unpruned recovery pool, and latency is the average post-failure generation time measured on an NVIDIA A100.

The shuffled-reference baseline follows the same pruning procedure but replaces the matched seed-0 failure trajectory with one from another request. The generic early-score baseline uses early candidate information without the request-specific failure reference. The temporal-consistency variant applies the complete rule above.

The fixed-pool experiment isolates the quality of the pruning signal. In the end-to-end FARE pipeline, the same pruning rule is applied after failure-aware allocation. Under the default SD2.1 setting, $B = 9$ complete recovery trajectories require

$$9 \times 50 = 450 \tag{49}$$

post-failure denoising steps. FARE without pruning uses this complete budget and obtains 76.2% recovery on CompBench. With failure-referenced pruning, the realized average generation cost is reduced to approximately 350 denoising steps, while recovery is 75.8%.

In full-generation equivalents, these costs correspond to

$$450/50 = 9.0 \qquad \text{and} \qquad 350/50 = 7.0, \tag{50}$$

respectively. Thus, the pruning module reduces redundant denoising after allocation while leaving the semantic evaluation criterion unchanged.