SPARSEGRN: ACCELERATING GENERATIVE REFINEMENT NETWORKS WITH ADAPTIVE SPARSE QUERIES

Anonymous authors
Paper under double-blind review



Figure 1: Qualitative comparison of dense GRN (top in each pair) and ***SparseGRN*** (bottom) on 2B (top block) and 8B (bottom block) models. ***SparseGRN*** accelerates pre-trained GRNs via training-free adaptive sparse query routing with full key/value coverage; per-example speedups over dense GRN are annotated above each pair, with visual fidelity largely preserved at reduced latency.

ABSTRACT

Generative Refinement Networks (GRNs) synthesize images by repeatedly revisiting the complete token map, enabling global error correction but incurring substantial inference cost. Recomputing every visual token at every refinement step introduces redundancy as image content stabilizes. We present ***SparseGRN***, a training-free framework that accelerates pre-trained GRNs through refinement-aware adaptive query routing. Our analysis reveals that semantic structure forms before local detail refinement is complete, bit uncertainty identifies tokens that remain sensitive to refinement, and sparse query updates benefit from complete key/value coverage. Building on these observations, ***SparseGRN*** matches global uncertainty to offline dense reference statistics to determine a prompt-dependent routing onset. During routed refinement, a budget-adaptive controller combines prompt-conditioned response estimates with an offline dynamic-programming table to allocate queries across the remaining steps. High-uncertainty tokens receive fresh attention and feed-forward updates, while inactive positions reuse cached block outputs and keys and values are recomputed over the complete context. This design preserves GRN's sampling schedule and full-map sampling procedure while reducing per-step computation. On an NVIDIA A100 GPU, ***SparseGRN*** achieves a $1.46\times$ speedup, reducing latency from 20.96 s to 14.34 s with a GenEval score of about 0.77 and a DPG-Bench decrease of only 0.08 points.

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.

1 INTRODUCTION

Recent advances in diffusion (Chen et al., 2024; Esser et al., 2024) and autoregressive (AR) models (Sun et al., 2024; Tian et al., 2024; Han et al., 2025) have enabled high-quality image synthesis. Generative Refinement Networks (GRNs) (Han et al., 2026) introduce a new paradigm that combines Hierarchical Binary Quantization (HBQ) with global refinement. Starting from a random token map, GRN repeatedly predicts the complete map, allowing previously generated content to be revised as context improves. Although GRN also supports complexity-aware step selection, its standard benchmark configuration uses 50 refinement steps, each processing the full token map. This repeated dense computation makes high-quality generation time-consuming, even after much of the image has stabilized.

Existing acceleration methods reduce sampling steps or reuse intermediate features in diffusion models (Salimans & Ho, 2022; Ma et al., 2024c), and exploit token redundancy or skip late scales in visual AR models (Guo et al., 2025; Li et al., 2025a; 2026). However, these designs target continuous denoising or next-scale generation, and do not directly account for GRN's stochastic full-map refinement: any position may change again, and stable regions can still provide context for correcting other tokens. Our analysis yields three observations. First, semantic structure forms before local detail refinement is complete, revealing redundancy in later steps (Fig. 2). Second, high-bit-uncertainty tokens are more sensitive to suppressing noise-based refinement, suggesting that they should receive greater update priority (Fig. 3). Third, sparse queries preserve image content more effectively than sparse keys and values, highlighting the importance of complete contextual coverage (Fig. 4). Together, these findings motivate selective computation that retains global context and the opportunity for continued refinement.

Based on these observations, we propose ***SparseGRN***, a training-free framework for accelerating pre-trained GRNs through refinement-aware adaptive query routing (Fig. 5). ***SparseGRN*** first matches global bit uncertainty against offline dense reference statistics to determine a prompt-dependent routing onset, allowing slower-converging prompts to retain dense computation for longer. After routing begins, a budget-adaptive controller distributes queries across the remaining steps instead of assigning a fixed ratio to every step. At the spatial level, high-uncertainty tokens receive fresh attention and feed-forward updates, while inactive positions reuse cached block outputs. Keys and values are recomputed from the complete assembled block input to preserve contextual coverage. By separating compute routing from sampling-state updates, ***SparseGRN*** reduces per-step computation while retaining GRN's noise-mixing schedule and full-map sampling procedure.

Experiments on GenEval, DPG-Bench, COCO2014, and COCO2017 demonstrate that ***SparseGRN*** accelerates generation while largely preserving quality. On an NVIDIA A100 GPU, our default configuration achieves a $1.46\times$ speedup. These results support refinement-aware routing as an effective quality–efficiency trade-off. Ablations show that combining adaptive onset and dynamic allocation improves GenEval by 0.016 and DPG by 0.182 points over the fixed-onset, constant-15% reference at comparable latency.

To summarize, our main contributions are:

- **Analysis of GRN refinement.** We identify late-stage computational redundancy, uncertainty-dependent refinement sensitivity, and the importance of complete KV coverage, establishing the basis for selective query updates.
- **Refinement-aware adaptive acceleration.** We introduce ***SparseGRN***, which combines prompt-dependent routing onset, budget-adaptive query allocation, and uncertainty-guided sparse updates with full KV coverage, without additional model training or changes to the sampling rules.
- **Multi-benchmark evaluation.** We demonstrate a $1.46\times$ speedup on the 2B GRN while largely retaining text–image alignment, distributional quality, and preference-related scores across complementary benchmarks.

2 RELATED WORK

Autoregressive Models. Autoregressive (AR) image generation typically factorizes the joint distribution of discrete visual tokens. Next-token methods (Razavi et al., 2019; Lee et al., 2022; Yu

et al., 2022; Sun et al., 2024) and masked models like MaskGIT (Chang et al., 2022) generate tokens causally or in parallel, while Visual Autoregressive (VAR) models (Tian et al., 2024; Tang et al., 2025; Ma et al., 2024a; Wang et al., 2025b; Han et al., 2025) adopt a coarse-to-fine next-scale prediction paradigm. However, these approaches generally freeze earlier generated tokens, limiting their ability to revisit and correct previous errors. In contrast, Generative Refinement Networks (GRNs) (Han et al., 2026) enable global revision by repeatedly predicting a fixed-resolution token map, using stochastic updates to refine content. While this full-map refinement is powerful, it introduces computational redundancy at late stages. ***SparseGRN*** aims to reduce this per-step cost through adaptive query routing while fully preserving GRN's exact sampling rules.

Efficient Visual Generation. Accelerating visual generation generally involves reducing the number of model evaluations or their per-step cost. Diffusion acceleration relies heavily on distillation (Meng et al., 2023; Salimans & Ho, 2022), quantization (Shang et al., 2023; Li et al., 2023b), and feature caching (Li et al., 2023a; Liu et al., 2024; Ma et al., 2024b;c). For AR models, recent works address causal decoding bottlenecks via speculative execution and parallelization (Wang et al., 2025a; Teng et al., 2025; Jang et al., 2025). In next-scale generation, various methods exploit stage-specific redundancy through token pruning, scale skipping, and attention compression (Guo et al., 2025; Chen et al., 2025; Li et al., 2025a; Xie et al., 2024; Li et al., 2025b; 2026). However, these techniques are tailored for continuous denoising or hierarchical scale prediction. They do not directly translate to GRN's stochastic full-map refinement, where stable regions must be retained to provide critical global context for correcting active tokens. ***SparseGRN*** addresses this unique paradigm by separating compute routing from sampling-state updates, utilizing bit uncertainty to guide sparse query updates with complete key/value coverage.

3 METHOD

3.1 PRELIMINARIES

Binary image representation. GRN (Han et al., 2026) represents an image using a Hierarchical Binary Quantization (HBQ) tokenizer. Each of the C latent channels is encoded with M coarse-to-fine bits, giving a binary map $Y \in \{0,1\}^{L\times d}$, where $L = HW$ is the number of spatial positions and $d = CM$ is the number of bits per position. Our backbone uses $C = 64$ and $M = 4$, so $d = 256$. The multi-token head predicts all binary coordinates in parallel.

Full-map prediction. GRN initializes its state as $F_0 = Y_{\text{rand}}$, a random binary map sampled once and reused throughout refinement. At step t, the state mixes the latest prediction with this fixed map:

$$F_t = S_t \odot Y_t + (1 - S_t) \odot Y_{\text{rand}}, \tag{1}$$

where Y_t is the latest predicted label map, $S_t \in \{0,1\}^{L\times d}$ is a binary selection map, and $\odot$ denotes element-wise multiplication. Selected coordinates retain predicted bits; the others take their values from Y_{rand}. The binary state is embedded into spatial feature tokens before prediction:

$$\widetilde{X}_t = \mathcal{E}(F_t) \in \mathbb{R}^{H\times W\times D}, \qquad X_t = \text{Flatten}_{HW}(\widetilde{X}_t) \in \mathbb{R}^{L\times D}, \tag{2}$$

where $\mathcal{E}$ applies one-hot encoding followed by a learned projection to each d-bit vector, D is the feature width, and Flatten_{HW} flattens the spatial dimensions. Attention and caching therefore operate on L feature tokens rather than individual bits. The Transformer then predicts the full label distribution and samples the next map:

$$p_\theta(Y_{t+1} \mid F_t, c) = \Phi_\theta(X_t, c), \qquad Y_{t+1} \sim p_\theta(\cdot \mid F_t, c), \tag{3}$$

where Φ_θ is the pre-trained GRN Transformer with its multi-token prediction head, and c denotes the class or text conditioning embedding.

Stochastic state updates. A retention schedule controls how much of the sampled prediction enters the next state. GRN independently samples $\xi_{t+1,i,j} \sim \text{Uniform}(0,1)$ for each token i and bit j, and sets $S_{t+1,i,j} = 1$ iff $\xi_{t+1,i,j} < l_{t+1}$, where l_{t+1} is the expected fraction of retained predicted bits. It then updates $F_{t+1} = S_{t+1} \odot Y_{t+1} + (1 - S_{t+1}) \odot Y_{\text{rand}}$. As l_t rises from 0 to 1, the state moves from random bits toward a complete image. The selection map is resampled at each step, so a coordinate may return to its fixed random value and receive a new prediction later. Previously

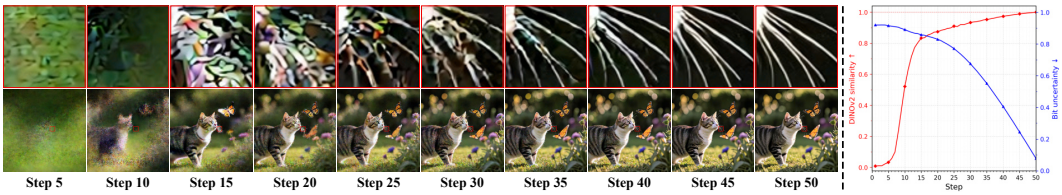


Figure 2: GRN refinement dynamics. ***Left:*** Refinement progress with enlarged details. ***Right:*** Average DINOv2 similarity (red) and bit uncertainty (blue) on COCO2014.

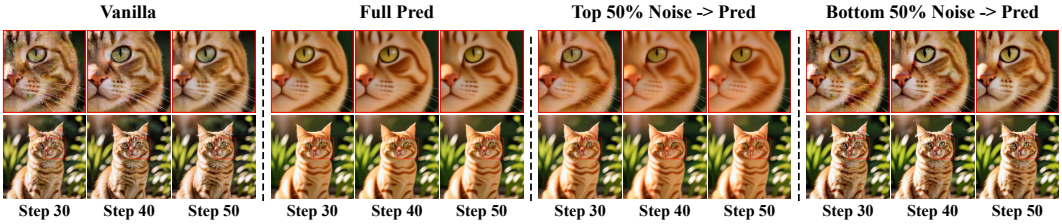


Figure 3: Effect of suppressing noise-based refinement.

generated coordinates are therefore revisable rather than frozen. Training reconstructs complete label maps from mixtures of ground-truth and Inference uses multiple refinement steps (e.g., $T = 50$) until $l_T = 1$, after which the HBQ decoder reconstructs the image at a high computational cost.

3.2 Observations

We aim to accelerate GRN inference while retaining its noise-mixing schedule and full-map sampling procedure. We therefore ask three questions: *when* can computation be reduced, *which* visual tokens still require updates, and *what* context must those updates retain? ***Observations 1 and 2*** reveal acceleration potential across refinement steps and tokens, respectively. ***Observation 3*** then identifies how to exploit this potential without compromising quality.

Observation 1: Semantic formation reveals when acceleration becomes possible. In Fig. 2, outputs sampled every 5 steps along the 50-step trajectory show a recognizable object and a well-established layout by around step 20. Subsequent steps mainly refine local details, as illustrated by the enlarged views of the cat's whiskers. The dense reference statistics associate lower bit uncertainty with greater semantic convergence. For 300 prompts from COCO 2014, Fig. 2 reports DINOv2 feature similarity between each intermediate image and the step-50 output of its own dense trajectory, averaged over the 300 samples, alongside mean bit uncertainty. From step 20 onward, similarity increases toward 1 as uncertainty decreases.

These trends identify *when* acceleration becomes plausible: after the main semantic structure has formed, around step 20 in the observed trajectories. The remaining steps offer an opportunity to reduce per-step computation rather than remove sampling transitions.

Observation 2: Bit uncertainty suggests which tokens to prioritize. We explore token-wise refinement sensitivity by selectively suppressing noise in the GRN state. We use bit-level predictive uncertainty to assess which spatial tokens may still require intensive refinement. Specifically, we compute token-level and global uncertainty as

$$u_{t,i} = \frac{1}{d}\sum_{j=1}^{d} 4p_{t,i,j}(1 - p_{t,i,j}), \qquad U_t = \frac{1}{L}\sum_{i=1}^{L} u_{t,i}, \tag{4}$$

where $p_{t,i,j}$ is the probability that bit j at visual token i equals one in the prediction of Y_t. The score $u_{t,i}$ averages normalized Bernoulli variances over the d bits, while U_t averages these scores over all L positions. Each bit's contribution is maximal at $p_{t,i,j} = 0.5$ and tends to zero as its prediction becomes more confident.

We intervene in the refinement process from step 25 onward by suppressing noise at selected positions. As shown in Fig. 3, applying this to high-uncertainty tokens degrades fidelity, whereas low-uncertainty tokens are robust to it. This indicates $u_{t,i}$ identifies tokens needing updates.

Observation 3: Sparse queries benefit from complete key/value coverage. Building on ***Observations 1 and 2***, we compare query and key/value sparsity to determine *what* context sparse updates must retain. Both interventions retain the top-50% visual positions ranked by bit uncertainty. As shown in Fig. 4, retaining only 50% of keys and values from step 25 onward leads to increasingly severe loss of visual information as refinement proceeds, whereas retaining 50% of queries preserves the main image content. Unlike sparse queries which preserve main content (Fig. 4 (top)), dropping key/value pairs destroys essential global context from stable regions, impairing subsequent refinement (Fig. 4 (bottom)).

The comparison motivates an attention pattern: sparse visual queries with complete key/value coverage. We project keys and values from the assembled input to each block. Writing Z for this full block input, including auxiliary context tokens and omitting normalization, an active visual set A_t uses $Q_{A_t} = Z_{A_t} W_Q$, $K_{\text{full}} = ZW_K$, and $V_{\text{full}} = ZW_V$, followed by $\text{Attn}(Q_{A_t}, K_{\text{full}}, V_{\text{full}})$. Only active queries receive new attention and feed-forward updates, while keys and values are recomputed for the complete context in every block.



Figure 4: Query versus key/value sparsity. ***Top:*** Sparse queries with full KV coverage. ***Bottom:*** Sparse keys and values.

In summary, ***Observations 1 and 2*** establish the potential for late-stage, uncertainty-guided acceleration; ***Observation 3*** provides the implementation direction: sparse query updates with complete key/value coverage. Together, they motivate the refinement-aware adaptive query routing described in Section 3.3, which approximates per-step predictions while retaining GRN's sampling schedule and full-map sampling procedure.

3.3 Refinement-Aware Adaptive Query Routing

Building on our observations, ***SparseGRN*** introduces a training-free routing mechanism (Fig. 5). A refinement-aware onset gate determines when sparse computation begins, while a budget-adaptive router dynamically allocates updates to high-uncertainty tokens using complete KV coverage.

Refinement-Aware Routing Onset. The onset gate uses uncertainty to estimate whether a prompt has reached a sufficiently mature refinement stage. Motivated by ***Observation 1***, we match its uncertainty to a dense reference trajectory, allowing prompts to reach this stage at different steps. We index refinement steps by $t = 1, \ldots, T$, with step t predicting Y_t from F_{t-1} in Eq. (3); routing therefore uses the preceding prediction's U_{t-1} and $u_{t-1,i}$. We estimate the prompt's *effective refinement age* by matching its preceding uncertainty to offline dense reference statistics:

$$a_t = \underset{r \in \mathcal{R}}{\text{argmin}} \left(\frac{\log U_{t-1} - \mu_r}{\max(\sigma_r, \epsilon)} \right)^2, \tag{5}$$

where $\mathcal{R}$ is the set of reference steps, μ_r and σ_r are the mean and standard deviation of $\log U_{r-1}$ across dense reference prompts, and $\epsilon > 0$ prevents division by a near-zero standard deviation. The selected stage a_t indicates which reference uncertainty best matches the current prompt. We standardize the log-uncertainty against reference statistics

Sparse routing begins after persistent evidence of maturity or when the dense-phase limit is reached. The maturity test requires $a_t \geq \tau$ for consecutive monitored steps, where τ is a calibrated threshold; a latest-onset bound caps the dense phase. Denoting the resulting onset by s, steps before s remain dense, and steps from s onward use the query router. The reference statistics, calibration, and persistence rule are specified in Section A.1. The gate requires neither an additional dense run of the test prompt nor online DINOv2 evaluation.

Budget-Adaptive Query Routing. At each sparse step, the router assigns fresh computation to the most uncertain visual positions. Following ***Observation 2***, we select the active set as

$$A_t = \text{TopK}\left(\{u_{t-1,i}\}_{i=1}^{L}, n(q_t) \right), \tag{6}$$

where $q_t \in \mathcal{Q} = \{0.05, 0.10, \ldots, 1.00\}$ is the visual-query fraction, $n(q_t) = \text{round}(Lq)$ is the corresponding token count, and TopK returns the indices of the $n(q_t)$ largest scores.

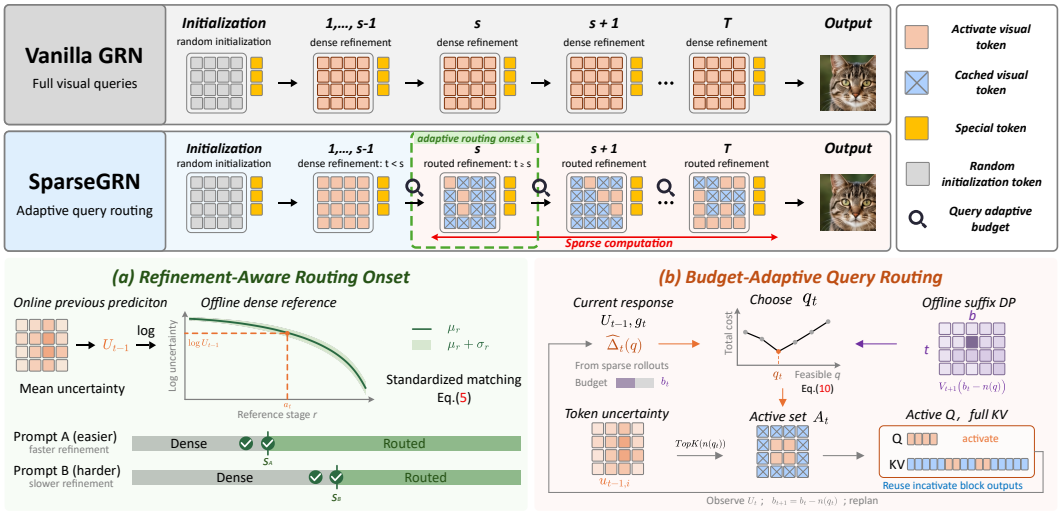


Figure 5: Overview of ***SparseGRN***. ***(a) Routing onset:*** Uncertainty matching and persistent maturity checks determine when sparse updates begin. ***(b) Query routing:*** Local responses and DP future costs allocate updates under a budget.

Sparse updates with complete context. As formulated in ***Observation 3***, active queries A_t receive fresh feed-forward and attention updates with full KV coverage recomputed from the complete context. Inactive positions simply reuse their same-block cached outputs initialized during the dense phase, which are overwritten whenever re-selected. Auxiliary conditioning queries remain fully active and are excluded from the visual budget.

Routing approximates predictions while retaining GRN's full-map sampling procedure. The unchanged output head predicts bits at all positions from the resulting features; the original sampling and noise-mixing rules are applied for all T steps. Under sparse execution, $u_{t,i}$ and U_t are computed from these approximate probabilities; inactive scores may remain unchanged. The ranking is reconsidered at every step, so cached positions can be selected again as scores or q_t change.

We allocate a shared query budget across the remaining steps to minimize predicted terminal log uncertainty. Queries used now reduce the budget available for subsequent refinement, so each allocation must account for both its immediate effect and the remaining steps. A fixed ratio cannot adapt this tradeoff to the observed refinement state. Given an average visual-query cap $\rho \in [0.05, 1]$ over the sparse phase, we formulate the following proxy objective, termed *Terminal-U* and motivated by the uncertainty–convergence relationship in ***Observation 1***:

$$\min_{\{q_t\}_{t=s}^{T}} \widehat{\log U_T} \quad \text{subject to} \quad \sum_{t=s}^{T} n(q_t) \le B_s, \qquad q_t \in \mathcal{Q}, \tag{7}$$

where $B_s = (T - s + 1)n(\rho)$ is the total visual-query budget for steps $s, \ldots, T$, and $\widehat{\log U_T}$ denotes the predicted terminal log uncertainty. The constraint allows query ratios to vary across steps and permits unused budget when additional queries are not predicted to improve the objective. Terminal-U is a confidence-based planning surrogate, with its responses calibrated under sparse execution, rather than a direct perceptual-quality metric.

Offline response estimation. To evaluate candidate allocations without extra Transformer evaluations, we construct an offline action-response table. Specifically, we estimate the expected uncertainty reduction $\hat{\Delta}_t(q|U_{t-1}, g_t)$ under varying query ratios and refinement states via Gaussian-weighted local regression on offline sparse rollouts.

A prompt-conditioned local estimator adapts the offline responses to the current refinement state. For sparse candidates $q < 1$, we estimate $\widehat{\Delta}_t(q \mid U_{t-1}, g_t)$ by Gaussian-weighted local regression over records with similar steps, ratios, uncertainties, and slopes. This assumes that nearby refinement states have similar expected responses; the slope distinguishes ongoing uncertainty reduction from a temporary plateau. The full-query candidate uses a population dense prior, as specified in Section A.2.

Budget allocation by dynamic programming. Terminal log uncertainty equals its pre-routing value plus the cumulative per-step changes, and the observed changes telescope:

$$\sum_{t=s}^{T} \Delta_t(q_t) = \log U_T - \log U_{s-1}, \tag{8}$$

where $\Delta_t(q_t)$ is the observed change under the executed action and U_{s-1} is the pre-routing uncertainty. Since the latter is fixed, predicting terminal uncertainty amounts to estimating the cumulative remaining changes.

We make future planning tractable by replacing state-dependent increments with fixed population responses. We pool offline records across prompt states to obtain risk-adjusted population responses $\bar{\Delta}_t(q)$ for each step and ratio. Under this additive approximation, two histories reaching the same step t with the same remaining budget b have identical future costs and feasible actions. Thus, the optimal continuation depends only on (t, b), which gives the optimal substructure required for dynamic programming.

A dynamic program tabulated backward from $t = T$ over integer budgets minimizes the predicted cumulative change. Each decision trades its immediate response against the best continuation under the remaining budget, giving the Bellman recurrence

$$V_t(b) = \min_{\substack{q \in \mathcal{Q} \\ n(q) \le b}} \left\{ \bar{\Delta}_t(q) + V_{t+1}\big(b - n(q)\big) \right\}, \qquad V_{T+1}(b) = 0, \tag{9}$$

where b is the remaining visual-query budget and $V_t(b)$ is the minimum predicted cumulative change over steps $t, \dots, T$. The two terms account for the current response $\bar{\Delta}_t(q)$ and the optimal suffix after spending $n(q)$ query tokens. The terminal condition applies to $b \ge 0$, since no queries remain after step T; infeasible states have value $+\infty$, preventing a current action from consuming tokens required to complete later steps. Each decision thus reserves queries for steps with greater predicted benefit. This recursion is exact for the fixed additive model, which approximates GRN's state-dependent dynamics.

Online routing combines a prompt-conditioned current response with the precomputed population estimate of future costs. At step t, we select the query fraction as

$$q_t = \operatorname*{argmin}_{\substack{q \in \mathcal{Q} \\ n(q) \le b_t}} \left\{ \widehat{\Delta}_t(q \mid U_{t-1}, g_t) + V_{t+1}\big(b_t - n(q)\big) \right\}, \tag{10}$$

where b_t is the remaining budget and $\widehat{\Delta}_t$ is the current response estimate, conditioned on U_{t-1} and g_t for sparse actions $q < 1$. The full-query action $q = 1$ retains the population dense prior, while V_{t+1} supplies the future cost. We initialize $b_s = B_s$ and use $g_s = 0$ as the zero-slope initialization.

Replanning after each action provides feedback without branching over unobserved future states. We execute only the selected current action, update $b_{t+1} = b_t - n(q_t)$, observe the resulting U_t, and replan at the next step. The offline collection samples a finite set of sparse histories; conditioning on the observed uncertainty and slope provides a local approximation, rather than a complete description of all possible GRN states. The details are in Section A.3.

4 Experiments

4.1 Experimental Setup

We build ***SparseGRN*** on the pre-trained 2B text-to-image GRN (Han et al., 2026) without additional training. Both use 50 refinement steps with full-map sampling. We evaluate alignment on GenEval (Ghosh et al., 2024) (553 prompts, four images each) and DPG-Bench (Hu et al., 2024), FID/recall/coverage on COCO2014/2017, and preference quality with HPSv2.1, PickScore, and ImageReward. Latency is measured on A100 (2B) and H200 (8B) GPUs, with speedup relative to dense GRN on the same hardware; other models follow their cited sources. Our fixed-routing baseline uses 25 dense plus 25 sparse steps at 15% queries, while ***SparseGRN*** adapts onset and allocation under a sparse-phase cap of $\rho = 0.15$. The onset gate is calibrated on 300 COCO prompts (reference step 24, two maturity checks, $t_{\max} = 29$), updating high-uncertainty tokens with full KV coverage; details in Section A. For model scaling, we evaluate 8B GRN and ***SparseGRN*** with default 50-step, $\rho = 0.15$ routing on H200, reporting GenEval overall and DPG overall.

Table 1: Text-to-image generation on GenEval and DPG-Bench. Latencies measured on A100 (2B) and H200 (8B) GPUs; speedup relative to the dense baseline on the same hardware.

Model	#Speed↑	#Latency↓	#Param	#Data	GenEval↑				DPG↑		
					Two Obj.	Pos.	Col.	Attr.	Over.	Glob.	Rel.	Over.
Diffusion Models												
PixArt-α (Chen et al., 2024)	-	-	0.6B	N/A	0.50	0.08	0.07	0.48	74.97	82.57	71.11	
SD3 Medium (Esser et al., 2024)	-	4.4s	2B	N/A	0.74	0.34	0.36	0.62	87.90	80.70	84.08	
JanusFlow (Ma et al., 2025)	-	-	1.3B	N/A	0.59	0.53	0.42	0.63	87.03	89.79	80.09	
SD3.5-Large (Esser et al., 2024)	-	-	8B	N/A	0.89	0.34	0.47	0.71	-	-	-	
Lumina-Image-2.0 (Qin et al., 2025)	-	-	2.6B	111M	-	-	-	0.73	-	-	87.20	
Z-Image-Turbo (Cai et al., 2025)	-	-	6B	N/A	0.95	0.65	0.68	0.82	91.29	92.16	84.86	
Z-Image (Cai et al., 2025)	-	-	6B	N/A	0.94	0.62	0.77	0.84	93.39	92.22	88.14	
Autoregressive Models												
Emu3-Gen (Wang et al., 2024)	-	-	8B	N/A	0.71	0.17	0.21	0.54	-	-	81.60	
HART (Tang et al., 2025)	-	-	0.7B	N/A	0.50	0.16	0.82	0.48	81.76	87.50	74.76	
Infinity† (Han et al., 2025)	-	2.2s	2B	160M	0.85	0.49	0.57	0.73	85.10	92.37	83.12	
GRN (Han et al., 2026)	1.0×	20.96s (A100)	2B	80M	0.92	0.54	0.61	0.76	83.89	94.23	86.27	
SparseGRN (Ours)	1.46×	14.34s (A100)	2B	80M	0.92	0.56	0.63	0.77	82.98	94.13	86.19	
GRN (Han et al., 2026)	1.0×	9.25s (H200)	8B	80M	0.80	0.41	0.55	0.67	79.94	93.38	80.77	
SparseGRN (Ours)	1.43×	6.46s (H200)	8B	80M	0.82	0.42	0.59	0.70	79.81	93.22	80.65	

4.2 Main Results

Comparison with Baselines. Tab. 1 presents quantitative comparisons on GenEval and DPG-Bench. ***SparseGRN*** achieves a 1.46× speedup over dense GRN, reducing latency from 20.96 s to 14.34 s. At this operating point, the unrounded GenEval score changes from 0.76244 to 0.76834, while DPG decreases by 0.08 points, from 86.27 to 86.19. ***SparseGRN*** obtains DPG global and relation scores of 82.98 and 94.13, respectively. GenEval position and color-attribute scores change from 0.54 to 0.56 and from 0.61 to 0.63, respectively. These results demonstrate that adaptive query routing reduces generation time while largely retaining alignment on both compositional prompts and dense descriptions. As shown in the qualitative comparison in Fig. 1, ***SparseGRN*** preserves high visual quality, with generated results remaining visually consistent with those of vanilla GRN. The same routing generalizes to the 8B backbone under the default 50-step, $\rho = 0.15$ configuration on an H200 GPU: GenEval improves from 0.67 to 0.70 with a 1.43× speedup (9.25s to 6.46s).

Generation Quality on COCO. Tab. 2 further evaluates generation quality on COCO2014 and COCO2017. CLIP scores are computed using CLIP ViT-B/32. At 1.46× speedup, ***SparseGRN*** increases FID by 0.37 and 0.12, respectively, while improved recall remains close to dense GRN. Coverage stays around 0.65 on COCO2014 and changes from 0.83 to 0.85 on COCO2017. CLIP scores increase from 0.312 to 0.315 on both datasets. The results indicate that the acceleration largely preserves distributional quality and reference-set coverage.

Table 2: Generation quality on COCO2014 and COCO2017.

Methods	Speed↑	Recall↑	Cov.↑	FID↓	CLIP↑
GRN	1.0×	0.33	0.65	23.69	0.312
SparseGRN	1.46×	0.34	0.65	24.06	0.315
GRN	1.0×	0.43	0.83	33.59	0.312
SparseGRN	1.46×	0.43	0.85	33.71	0.315

Table 3: Automated preference evaluation.

Methods	Speed↑	Lat.↓	HPS↑	PS↑	IR↑
GRN	1.0×	20.96s	29.24	21.75	1.04
SparseGRN	1.46×	14.34s	29.01	21.73	1.03

Human-Preference Metrics. Tab. 3 reports automated preference metrics. ***SparseGRN*** obtains 29.01 HPSv2.1 (HPS), 21.73 PickScore (PS), and 1.03 ImageReward (IR), compared with 29.24, 21.75, and 1.04 for dense GRN. These small absolute changes complement the alignment and COCO evaluations, supporting the quality–efficiency trade-off of the default configuration.

4.3 Additional Analysis

We study adaptive onset, reference maturity, query allocation, few-step robustness, and token/output visualizations. Unless noted, variants use default $\rho = 0.15$ with adaptive onset and paired prompts/seeds. Speedups are relative to dense GRN; GenEval/DPG are overall scores, circled numbers index table rows, and all differences are absolute. Protocols in Section B.

Table 4: Component ablation at a 15% query cap. Rows 3 and 4 separately add adaptive onset and dynamic allocation to row 2.

Methods	Speed↑	Lat.↓	GenEval↑	DPG↑
① Vanilla GRN	1.0×	20.96s	0.762	86.271
② Fixed sparse	1.45×	14.42s	0.752	86.011
③ + Adaptive onset	1.47×	14.26s	0.757	86.074
④ + Dynamic alloc.	1.45×	14.47s	0.767	86.183
⑤ ***SparseGRN***	1.46×	14.34s	0.768	86.193

Table 5: Effect of the reference maturity step. All routed variants use two maturity checks and dynamic allocation at a 15% cap.

Ref	Latest s	Speed↑	GenEval↑	DPG↑
① 20	25	1.56×	0.722	85.563
② **24 (default)**	29	1.46×	0.768	86.193
③ 30	35	1.35×	0.766	86.441
④ 35	40	1.27×	0.762	86.616
⑤ Vanilla GRN	N/A	1.0×	0.762	86.271

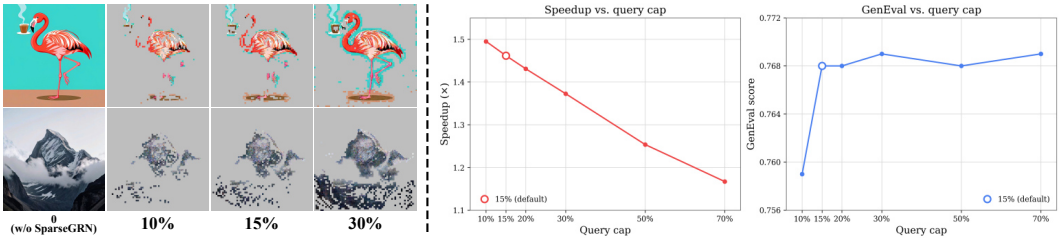


Figure 6: Active-token visualization and query-budget trade-off. ***Left:*** Dense references and active tokens at 10%, 15%, and 30% caps. ***Right:*** Speedup and GenEval versus query cap ρ under dynamic allocation. Highlighted markers denote the default 15% cap.

Impact of Adaptive Routing. In Tab. 4, ① is dense GRN and ② uses fixed $s = 26$ with constant 15% queries. Adding adaptive onset (③) improves GenEval by 0.005 and DPG by 0.063 points at 1.47× speedup. Dynamic allocation alone (④) improves the two scores by 0.015 and 0.172 at 1.45×. Combining both (⑤) yields the highest scores among sparse variants.

Different Reference Maturity Steps. Tab. 5 varies the reference over 20, 24, 30, and 35 with dynamic allocation at $\rho = 0.15$. Ref 20 is fastest (1.56×), but lowers GenEval to 0.722 and DPG to 85.563. Ref 24 achieves the highest GenEval (0.768) at 1.46×. Refs 30 and 35 increase DPG to 86.441 and 86.616, respectively, but reduce speedup to 1.35× and 1.27× without gains.

Visualization of Active Tokens. The left panel of Fig. 6 visualizes active tokens for a flamingo and a mountain scene at query caps of 10%, 15%, and 30%, alongside the dense GRN reference. Gray regions indicate unselected positions. The DP-based controller determines the query count, and bit-uncertainty Top-K selects the positions. Inactive tokens reuse cached outputs but remain eligible for later updates and participate in full-map sampling.

Different Active-Query Budgets. The right panels of Fig. 6 evaluate the quality–speed trade-off at $\rho \in \{0.10, 0.15, 0.20, 0.30, 0.50, 0.70\}$, keeping the adaptive onset gate fixed. At each cap, ***SparseGRN*** uses dynamic allocation under $B_s = (50 - s + 1)n(\rho)$. Here, ρ denotes the sparse-phase average activation cap, and speedup is measured relative to dense GRN. Across these settings, ***SparseGRN*** achieves speedups from 1.17× to 1.50× and GenEval scores from 0.759 to 0.769. The highlighted default 15% cap achieves a 1.46× speedup and a GenEval score of 0.768. This sweep extends Tab. 7 to a wider range of query budgets under the same adaptive onset gate.

5 Conclusion

This work presents ***SparseGRN***, a training-free framework for accelerating Generative Refinement Networks. We reveal that repeated dense computation introduces redundancy after semantic structure forms, while uncertain tokens still require updates with complete contextual coverage. Building on these observations, ***SparseGRN*** combines a prompt-dependent routing onset with DP-based adaptive query allocation. High-uncertainty tokens receive fresh attention and feed-forward updates, while inactive positions reuse cached block outputs and full KV coverage is retained. This design reduces per-step computation while preserving GRN's sampling schedule and full-map sampling procedure. Experiments on multiple benchmarks demonstrate a 1.46× speedup with largely preserved generation quality, supporting refinement-aware compute allocation as an effective approach to efficient GRN inference.

AI USE STATEMENT

In this work, we used generative AI tools to assist with manuscript editing. GPT-based tools were used to improve the language, grammar, and readability of the manuscript without changing the underlying scientific content. The underlying data and numerical results in the figure were not generated or modified by AI.

All AI-assisted outputs were manually reviewed and verified by the authors. The authors take full responsibility for the final manuscript, including all text, figures, experimental results, scientific claims, and conclusions.

REFERENCES

Huanqia Cai, Sihan Cao, Ruoyi Du, Peng Gao, Steven Hoi, Zhaohui Hou, Shijie Huang, Dengyang Jiang, Xin Jin, Liangchen Li, et al. Z-Image: An efficient image generation foundation model with single-stream diffusion transformer. *arXiv preprint arXiv:2511.22699*, 2025.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. MaskGIT: Masked generative image transformer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11315–11325, 2022.

Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, and Zhenguo Li. PixArt-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. In *International Conference on Learning Representations*, volume 2024, pp. 57611–57640, 2024.

Zhuokun Chen, Jugang Fan, Zhuowei Yu, Bohan Zhuang, and Mingkui Tan. Frequency-aware autoregressive modeling for efficient high-resolution image synthesis. In *Proceedings of the International Conference on Computer Vision*, 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 *International Conference on Machine Learning*, 2024.

Dhruba Ghosh, Hannaneh Hajishirzi, and Ludwig Schmidt. GenEval: An object-focused framework for evaluating text-to-image alignment. *Advances in Neural Information Processing Systems*, 36, 2024.

Hang Guo, Yawei Li, Taolin Zhang, Jiangshan Wang, Tao Dai, Shu-Tao Xia, and Luca Benini. FastVAR: Linear visual autoregressive modeling via cached token pruning. In *Proceedings of the International Conference on Computer Vision*, pp. 19011–19021, 2025.

Jian Han, Jinlai Liu, Yi Jiang, Bin Yan, Yuqi Zhang, Zehuan Yuan, Bingyue Peng, and Xiaobing Liu. Infinity: Scaling bitwise autoregressive modeling for high-resolution image synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2025.

Jian Han, Jinlai Liu, Jiahuan Wang, Bingyue Peng, and Zehuan Yuan. Generative refinement networks for visual synthesis. In *European Conference on Computer Vision*, 2026.

Xiwei Hu, Rui Wang, Yixiao Fang, Bin Fu, Pei Cheng, and Gang Yu. ELLA: Equip diffusion models with LLM for enhanced semantic alignment. *arXiv preprint arXiv:2403.05135*, 2024.

Doohyuk Jang, Sihwan Park, June Yong Yang, Yeonsung Jung, Jihun Yun, Souvik Kundu, Sung-Yub Kim, and Eunho Yang. LANTERN: Accelerating visual autoregressive models with relaxed speculative decoding. In *International Conference on Learning Representations*, 2025.

Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11523–11532, 2022.

Jiajun Li, Yue Ma, Xinyu Zhang, Qingyan Wei, Songhua Liu, and Linfeng Zhang. SkipVAR: Accelerating visual autoregressive modeling via adaptive frequency-aware skipping. *arXiv preprint arXiv:2506.08908*, 2025a.

Kunjun Li, Zigeng Chen, Cheng-Yen Yang, and Jenq-Neng Hwang. Memory-efficient visual autoregressive modeling with scale-aware KV cache compression. In *Advances in Neural Information Processing Systems*, 2025b.

Senmao Li, Taihang Hu, Fahad Shahbaz Khan, Linxuan Li, Shiqi Yang, Yaxing Wang, Ming-Ming Cheng, and Jian Yang. Faster diffusion: Rethinking the role of U-Net encoder in diffusion models. *arXiv preprint arXiv:2312.09608*, 3:1, 2023a.

Senmao Li, Kai Wang, Salman Khan, Fahad Shahbaz Khan, Jian Yang, and Yaxing Wang. FasterVAR: Plug-and-play acceleration for visual autoregressive models. In *International Conference on Machine Learning*, 2026.

Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-Diffusion: Quantizing diffusion models. In *Proceedings of the International Conference on Computer Vision*, pp. 17489–17499, 2023b.

Haozhe Liu, Wentian Zhang, Jinheng Xie, Francesco Faccio, Mengmeng Xu, Tao Xiang, Mike Zheng Shou, Juan-Manuel Perez-Rua, and Jürgen Schmidhuber. Faster diffusion via temporal attention decomposition. *arXiv preprint arXiv:2404.02747*, 2024.

Xiaoxiao Ma, Mohan Zhou, Tao Liang, Yalong Bai, Tiejun Zhao, Huaian Chen, and Yi Jin. STAR: Scale-wise text-to-image generation via auto-regressive representations. *arXiv preprint arXiv:2406.10797*, 2024a.

Xinyin Ma, Gongfan Fang, Michael Bi Mi, and Xinchao Wang. Learning-to-cache: Accelerating diffusion transformer via layer caching. *Advances in Neural Information Processing Systems*, 37: 133282–133304, 2024b.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. DeepCache: Accelerating diffusion models for free. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 15762–15772, 2024c.

Yiyang Ma, Xingchao Liu, Xiaokang Chen, Wen Liu, Chengyue Wu, Zhiyu Wu, Zizheng Pan, Zhenda Xie, Haowei Zhang, Xingkai Yu, et al. JanusFlow: Harmonizing autoregression and rectified flow for unified multimodal understanding and generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7739–7751, 2025.

Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 14297–14306, 2023.

Qi Qin, Le Zhuo, Yi Xin, Ruoyi Du, Zhen Li, Bin Fu, Yiting Lu, Jiakang Yuan, Xinyue Li, Dongyang Liu, et al. Lumina-Image 2.0: A unified and efficient image generative framework. *arXiv preprint arXiv:2503.21758*, 2025.

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with VQ-VAE-2. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

Yuzhang Shang, Zhihang Yuan, Bin Xie, Bingzhe Wu, and Yan Yan. Post-training quantization on diffusion models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1972–1981, 2023.

Peize Sun, Yi Jiang, Shoufa Chen, Shilong Zhang, Bingyue Peng, Ping Luo, and Zehuan Yuan. Autoregressive model beats diffusion: Llama for scalable image generation. *arXiv preprint arXiv:2406.06525*, 2024.

Haotian Tang, Yecheng Wu, Shang Yang, Enze Xie, Junsong Chen, Junyu Chen, Zhuoyang Zhang, Han Cai, Yao Lu, and Song Han. HART: Efficient visual generation with hybrid autoregressive transformer. In *International Conference on Learning Representations*, 2025.

Yao Teng, Han Shi, Xian Liu, Xuefei Ning, Guohao Dai, Yu Wang, Zhenguo Li, and Xihui Liu. Accelerating auto-regressive text-to-image generation with training-free speculative jacobi decoding. In *International Conference on Learning Representations*, 2025.

Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. In *Advances in Neural Information Processing Systems*, 2024.

Junke Wang, Zhi Tian, Xun Wang, Xinyu Zhang, Weilin Huang, Zuxuan Wu, and Yu-Gang Jiang. SimpleAR: Pushing the frontier of autoregressive visual generation through pretraining, SFT, and RL. *arXiv preprint arXiv:2504.11455*, 2025a.

Tong Wang, Guanyu Yang, Nian Liu, Kai Wang, Yaxing Wang, Abdelrahman M. Shaker, Salman Khan, Fahad Shahbaz Khan, and Senmao Li. Diversity has always been there in your visual autoregressive models. *arXiv preprint arXiv:2511.17074*, 2025b.

Xinlong Wang, Xiaosong Zhang, Zhengxiong Luo, Quan Sun, Yufeng Cui, Jinsheng Wang, Fan Zhang, Yueze Wang, Zhen Li, Qiying Yu, et al. Emu3: Next-token prediction is all you need. *arXiv preprint arXiv:2409.18869*, 2024.

Rui Xie, Tianchen Zhao, Zhihang Yuan, Rui Wan, Wenxi Gao, Zhenhua Zhu, Xuefei Ning, and Yu Wang. LiteVAR: Compressing visual autoregressive modelling with efficient attention and quantization. *arXiv preprint arXiv:2411.17178*, 2024.

Jiahui Yu, Yuanzhong Xu, Jing Yu Koh, Thang Luong, Gunjan Baid, Zirui Wang, Vijay Vasudevan, Alexander Ku, Yinfei Yang, Burcu Karagol Ayan, et al. Scaling autoregressive models for content-rich text-to-image generation. *arXiv preprint arXiv:2206.10789*, 2022.

A Adaptive Query Routing Details

Default Configuration and Budget. We use 1-based step indexing with $T = 50$. The fixed-routing baseline executes steps 1–25 densely and steps 26–50 with a constant 15% active visual-query ratio, corresponding to onset $s = 26$. ***SparseGRN*** adapts both this onset and the subsequent allocation. Given onset s, its default sparse-phase budget is $B_s = (50 - s + 1)n(0.15)$, where $n(q) = \mathrm{round}(Lq)$. The controller selects per-step ratios from $\mathcal{Q} = \{0.05, 0.10, \ldots, 1.00\}$, allowing individual steps to use more or fewer than 15% queries while respecting the remaining budget. The cap applies only to the sparse phase: a later onset retains more dense steps and therefore permits more total trajectory computation. At each routed step, the most uncertain tokens receive fresh updates with complete KV coverage. The onset rule below determines when this phase begins.

A.1 Reference Calibration and Routing Onset

We construct the dense reference in Eq. (5) by running GRN on 300 COCO reference prompts. For each reference step $r \in \mathcal{R} = \{20, \ldots, 41\}$, we record the mean μ_r and standard deviation σ_r of $\log U_{r-1}$. Throughout, uncertainty is clamped to a small positive value before taking logarithms, and $\epsilon > 0$ provides a numerical floor for the standard deviation. The resulting a_t identifies the closest reference stage; it does not estimate an absolute perceptual score.

A persistence check and a latest-onset bound jointly determine when sparse routing begins. Monitoring starts at $t_{\min} = 20$. We trigger routing when $a_t \geq \tau$ holds for two consecutive monitored steps, resetting the counter whenever the condition fails. The first step satisfying this persistence rule is the onset s. For reference prediction step k, we set $t_{\max} = k + 5$. If no earlier trigger occurs, routing starts at $t_{\max}$ to cap the dense phase, even if the maturity condition is unmet. Thus, the earliest possible onset is step 21. The default maturity reference is prediction step 24, with latest onset $t_{\max} = 29$. Since reference step r stores statistics of U_{r-1}, this corresponds to $\tau = 25$ in the step-indexed matching rule. If maturity is first reached after prediction step 24 and persists at the next check, steps 25 and 26 satisfy the test and routing starts at $s = 26$. This aligns the reference setting with the fixed-onset baseline without fixing every prompt's onset. Unless the onset configuration is explicitly varied, the gate parameters are fixed across comparisons.

A.2 Offline Action-Response Construction

Calibration records. Sparse calibration rollouts measure how query allocations change uncertainty under the same execution mechanism used online. The collection combines different onset schedules and fixed query ratios. At each recorded step, the selected queries are updated with complete KV coverage and inactive positions reuse their same-block outputs, while the output head, sampling procedure, and noise-mixing schedule are retained. Thus, the input states include the effects of preceding sparse updates. Each record is $r_k = (t_k, q_k, \ell_k, g_k, \delta_k)$, where ℓ_k is the log uncertainty before the action, g_k is the preceding one-step decrease in mean uncertainty, and δ_k is the observed log-uncertainty change after the action. In the step indexing of Section 3.3, these correspond to $\ell_k = \log U_{t_k-1}$, $g_k = U_{t_k-2} - U_{t_k-1}$, and $\delta_k = \log U_{t_k} - \log U_{t_k-1}$ along the recorded rollout.

The calibration data cover late-refinement actions and include additional interventions for early onset. The late-stage lookup uses four calibration prompts, combining measured sparse ratios at 5-percentage-point intervals to estimate responses for steps 26–50. Additional early-step interventions cover steps 20–25 to support onset before step 26. The COCO-300 dense runs provide the dense reference and population dense prior; the sparse responses come from separate calibration rollouts, rather than an exhaustive ratio sweep over all 300 prompts. The resulting data support the target steps and action grid while sampling only a finite set of preceding schedules and prompt states.

Local response estimator. Gaussian kernel weighting estimates each response from nearby actions and refinement states. For a candidate q at step t, the distance to calibration record k is

$$d_k^2(t,q) = \left(\frac{t_k - t}{h_t}\right)^2 + \left(\frac{q_k - q}{h_q}\right)^2 + \left(\frac{\ell_k - \log U_{t-1}}{h_\ell}\right)^2 + \left(\frac{g_k - g_t}{h_g}\right)^2, \tag{11}$$

where (h_t, h_q, h_ℓ, h_g) are bandwidths for the step, query ratio, log uncertainty, and slope, respectively. We set them to $(2.5, 0.06, 0.22, 0.025)$, with query ratios expressed as fractions. The step and ratio terms pool nearby interventions, while uncertainty and slope condition the estimate on the current refinement state. Using Gaussian weights $w_k = \exp(-d_k^2/2)$, we compute the response moments as

$$m_t(q) = \frac{\sum_k w_k \delta_k}{\sum_k w_k}, \qquad \nu_t^2(q) = \frac{\sum_k w_k (\delta_k - m_t(q))^2}{\sum_k w_k}, \tag{12}$$

where $m_t(q)$ and $\nu_t(q)$ are the weighted mean and standard deviation of the observed log-uncertainty changes δ_k.

The online estimator uses an explicit slope initialization and a population fallback. We set $g_s = 0$ at onset; for $t > s$ we use $g_t = U_{t-2} - U_{t-1}$ from successive observations. Offline records retain their measured slopes. Kernel weights no greater than 10^{-10} are discarded; if none remain, the estimator falls back to the population moments for that step and ratio.

We regularize response estimates with a variance penalty and a constrained action curve. We first form

$$\widetilde{\Delta}_t(q) = m_t(q) + \lambda \nu_t(q), \qquad \lambda = 0.15, \tag{13}$$

where λ controls the penalty on response variability. A reduction in uncertainty has a negative response, so the positive penalty makes high-variance actions less favorable. The coefficient is a calibration parameter, not a confidence level. We then take a running minimum over ascending query ratios, imposing a non-increasing predicted change as more queries are updated, and bound sparse responses from below by the dense response. These are regularizing assumptions on the estimated action curve, rather than pointwise guarantees about actual transitions. The resulting estimate is $\widehat{\Delta}_t(q \mid U_{t-1}, g_t)$.

Population response for future steps. Fixed population responses provide a reusable estimate of future refinement costs. We obtain population moments by pooling the same sparse records using only the step and ratio terms of Eq. (11). Applying the same variance penalty and action-curve regularization gives $\bar{\Delta}_t(q)$. The full-query action $q = 1$ uses the log-uncertainty transition of the population dense prior in both online and future estimates; this prior also supplies the bound for sparse actions. It serves as a regularization reference: the response to $q = 1$ after a sparse history can differ from a fully dense trajectory. These future responses are frozen before constructing the value table. Only the current sparse-action estimates are subsequently conditioned on the observed prompt, so the controller does not need to predict a separate future uncertainty trajectory for every candidate action sequence.

A.3 Dynamic Programming and Online Budget Updates

The budget $B_s = (T - s + 1)n(\rho)$ limits total visual-query tokens over steps $s, \ldots, T$. For example, with $T = 50$, $s = 26$, and $\rho = 0.15$, the remaining 25 steps receive a budget corresponding to an average of approximately 15% active visual queries. Individual ratios can differ from this average. Counting each visual position once per step, total visual-query use is $(s-1)L + \sum_{t=s}^{T} n(q_t)$; full-KV projections, auxiliary queries, and the output head incur additional costs, so ρ is not a full-trajectory FLOP ratio.

Backward tabulation enforces the query budget while reserving enough tokens for every remaining step. The dynamic program uses the fixed $\bar{\Delta}_t(q)$ and integer cost $n(q)$. For $q_{\min} = 0.05$, a state is infeasible when $b < (T - t + 1)n(q_{\min})$, because every remaining step must execute at least one allowed action. We initialize $V_{T+1}(b) = 0$ for nonnegative budgets and assign $+\infty$ to infeasible states. Proceeding backward from step T, we evaluate each feasible action at each budget using Eq. (9), storing its minimum value, selected action, and total token use. Once the row for $t + 1$ is available, the entire row for t follows from these lookups.

Optimal substructure follows from the fixed response approximation. After choosing q, any feasible continuation with cost greater than $V_{t+1}(b - n(q))$ can be replaced by the minimizing suffix without changing the current response or violating the budget. Repeated subproblems therefore share a single table entry. For a horizon of N steps and maximum integer budget B, tabulation requires $O(NB|\mathcal{Q}|)$ operations and $O(NB)$ storage, avoiding enumeration of $|\mathcal{Q}|^N$ complete schedules.

The compiled table can be reused across prompts and onset times covered by its step and budget ranges.

The online controller maintains budget feasibility by selecting one action and replanning after each observation. Each candidate in Eq. (10) requires one suffix lookup in addition to its current-state response estimate. Candidates with infeasible suffix values are discarded. Ties favor plans using fewer total query tokens, including both the current action and its selected suffix. After executing q_t, we set $b_{t+1} = b_t - n(q_t)$ and use the observed U_t to condition the next decision. Only this first action is committed; subsequent choices are recomputed from the new observation and remaining budget.

B Ablation Protocols

Shared Evaluation Settings. All ablations retain the 50-step sampling schedule, full-map prediction and sampling, full KV coverage, and inactive-output caching. Calibration prompts are separated from evaluation prompts, and comparisons use paired prompts and generation seeds. GenEval overall is the average of its six task scores. Latency includes the routing computations used by each evaluated variant but excludes external quality evaluation; speedup is the paired dense baseline's mean latency divided by the variant's mean latency under the same timing boundaries. The reference uses steps 1–25 densely and steps 26–50 with a constant 15% visual-query ratio.

Routing Onset and Budget Allocation. The four sparse rows of Tab. 4 form a 2×2 comparison of fixed/adaptive onset and constant/dynamic allocation. The reference sweep in Tab. 5 varies the maturity reference over 20, 24, 30, and 35 with latest-onset bounds 25, 29, 35, and 40, respectively. All routed variants retain the full dynamic controller; vanilla GRN provides the dense control. Each sparse configuration uses the same sparse-phase cap $\rho = 0.15$, but total visual-query use is $(s-1)L + \sum_{t=s}^{50} n(q_t)$ and therefore depends on onset. We record onset distributions and the fraction of prompts reaching the latest-onset bound. For latency-matched analysis, caps are selected on calibration prompts and frozen before evaluation; the actual latency gap is reported for each feasible comparison.

For the controller ablation in Tab. 6, onset and budget cap are shared per sample. Constant allocation uses $q_t = \rho$. Local greedy minimizes only $\widehat{\Delta}_t(q \mid U_{t-1}, g_t)$, while reserving at least $(T-t)n(q_{\min})$ query tokens for the remaining steps. The complete controller includes the DP suffix cost. Realized query use is computed as $\bar{q} = [\sum_{t=s}^{T} n(q_t)]/[(T-s+1)L]$ per sample and then averaged. The budget sweep in Tab. 7 uses the same adaptive onset gate, with reference prediction step 24 and two consecutive maturity checks, and compares constant versus dynamic allocation at $\rho \in \{0.10, 0.15, 0.20\}$. The gate is frozen across budgets, so each prompt/seed shares the same dense prefix and onset s; only $B_s = (50 - s + 1)n(\rho)$ and the allocation policy vary. Rows ③ and ④ of Tab. 7 reuse the constant-allocation and full-controller configurations in rows ① and ③ of Tab. 6, respectively. They also correspond to the adaptive-onset-only and full variants in Tab. 4.

Active-Query Budget Curves. For Fig. 6, the adaptive onset gate is frozen at the default reference prediction step 24 with two consecutive maturity checks. We vary only $\rho \in \{0.10, 0.15, 0.20, 0.30, 0.50, 0.70\}$ under dynamic allocation, yielding six configurations. For each prompt/seed, all configurations share the dense prefix and onset; the remaining budget is $B_s = (50 - s + 1)n(\rho)$. Dynamic allocation uses the full local-response-plus-DP controller with the same response estimator, action grid, and spatial uncertainty ranking across caps. The DP table must cover all evaluated budgets.

Each configuration uses the same 553 GenEval prompts with four images per prompt. We plot task-averaged GenEval and speedup relative to dense GRN under identical timing settings, including the controller's actual online overhead. We also record realized query use because the dynamic controller may leave budget unused. The adaptive-onset 15% dynamic configuration is shared with Tabs. 4 and 6. The 10%, 15%, and 20% points use the same dynamic-allocation settings as Tab. 7, which additionally reports DPG and constant-allocation baselines. Prompt-level paired bootstrap can quantify uncertainty in the GenEval differences.

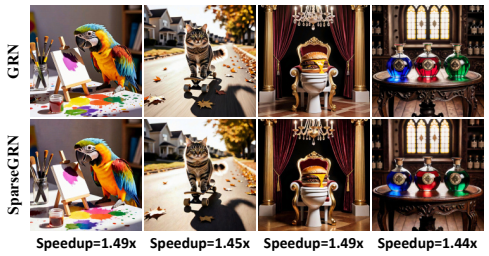


Figure 7: Qualitative comparison with detailed prompts. ***Top:*** Dense GRN. ***Bottom:*** ***SparseGRN*** with adaptive query routing. Per-example speedups are annotated below the pairs.

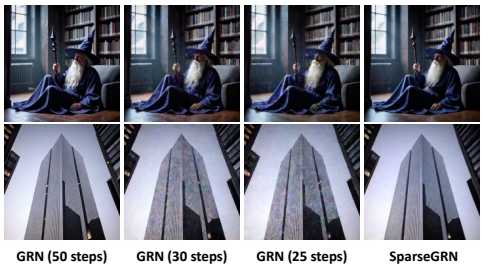


Figure 8: Few-step visualization: dense GRN with 50, 30, and 25 steps versus ***SparseGRN***. Fewer steps degrade dense GRN details, while ***SparseGRN*** preserves them.

C Additional Visual Results

Query Allocation. Tab. 6 compares constant allocation, greedy local responses, and local responses with DP planning at a 15% cap. At $1.46\times$ speedup, Local + DP improves GenEval by 0.011 over constant allocation and 0.056 over greedy allocation. The DPG gains are 0.119 and 0.741 points. Latencies remain similar (14.26–14.35 s), supporting accounting for future costs in allocation.

Table 6: Query-allocation ablation with adaptive onset and a 15% sparse-phase cap.

Policy	ρ	Speed↑	Lat.↓	GenEv.↑	DPG↑
① Constant	0.15	$1.47\times$	14.26s	0.757	86.074
② Greedy	0.15	$1.46\times$	14.35s	0.712	85.452
③ **Local + DP**	0.15	$1.46\times$	14.34s	0.768	86.193

Different Query Budgets. Tab. 7 compares constant and dynamic allocation at 10%, 15%, and 20% caps, sharing each sample's onset and budget $B_s = (50 - s + 1)n(\rho)$. Dynamic allocation improves GenEval by 0.017, 0.011, and 0.001, respectively, at $1.50\times$, $1.46\times$, and $1.43\times$ speedup. Larger gains at tighter caps suggest that redistributing limited budgets is useful.

Table 7: Effect of the query budget under shared adaptive onset.

Policy	ρ	Speed↑	Lat.↓	GenEv.↑	DPG↑
① Constant	0.10	$1.50\times$	13.97s	0.742	86.063
② Local + DP	0.10	$1.50\times$	14.02s	0.759	86.157
③ Constant	0.15	$1.47\times$	14.26s	0.757	86.074
④ **Local + DP**	0.15	$1.46\times$	14.34s	0.768	86.193
⑤ Constant	0.20	$1.41\times$	14.85s	0.767	86.157
⑥ Local + DP	0.20	$1.43\times$	14.65s	0.768	86.193

Qualitative Results with Detailed Prompts. Fig. 7 compares GRN and ***SparseGRN*** using four prompts depicting a painting parrot, a skateboarding cat, a cheeseburger in a royal chamber, and ornate potion bottles. Examples use our default routing configuration, with speedups varying across samples, and complement the quantitative alignment results. Full prompts appear in Section D; additional trajectories and comparisons appear in Figs. 9 and 10.

Few-Step Performance. Tab. 8 compares dense GRN with $T \in \{25, 30, 40, 50\}$ against ***SparseGRN*** at $T = 50$ under the default $\rho = 0.15$ routing (A100). Fewer dense steps steadily degrade alignment: GenEval overall drops $0.762 \to 0.746 \to 0.689 \to 0.648$ and DPG $86.27 \to 86.10 \to 85.82 \to 85.15$, with Fig. 8 showing the corresponding loss of fine local details. ***SparseGRN*** instead retains 0.768 GenEval and 86.19 DPG at $1.46\times$ speedup.

Table 8: Few-step robustness on GenEval overall and DPG overall. Dense GRN uses total steps $T \in \{25, 30, 40, 50\}$; ***SparseGRN*** uses default $\rho = 0.15$ adaptive routing at 50 steps.

Method	Steps	Speed↑	Lat.↓	GenEval↑	DPG↑
GRN	50	$1.0\times$	20.96s	0.762	86.27
GRN	40	$1.24\times$	16.90s	0.746	86.10
GRN	30	$1.60\times$	13.10s	0.689	85.82
GRN	25	$1.95\times$	10.75s	0.648	85.15
SparseGRN	50	$1.46\times$	14.34s	0.768	86.19

Visualization of Refinement Trajectories. Fig. 9 presents six dense GRN trajectories, covering a penguin, a snail, a squirrel, a raccoon, a waterfall, and a valley road. Columns show steps 5–50 at five-step intervals. Each example pairs full images with enlarged local regions above them. The subjects and layouts generally become recognizable before local textures stabilize; the enlarged views reveal continued changes in fur, whiskers, water, and road details. These examples comple-

ment ***Observation 1***, motivating selective computation in later refinement steps while retaining the opportunity for further updates. The corresponding prompts are listed in Section D.8.

Additional Qualitative Comparisons. Fig. 10 compares 24 paired outputs from dense GRN and ***SparseGRN***. The four groups cover people, animals, scenes and objects, and detailed long prompts. Within each group, dense GRN outputs appear above ***SparseGRN*** outputs. The pairs retain similar subjects and overall compositions, with some differences in local appearance, illustrating the qualitative quality–efficiency trade-off beyond Figs. 1 and 7. All examples use the default routing configuration, but runtimes vary by prompt: latencies range from 13.880 s to 14.767 s, corresponding to $1.42\times$–$1.51\times$ speedups against the 20.96 s dense baseline. These are per-example measurements rather than the aggregate $1.46\times$ speedup reported in the main experiments. Full prompts, latencies, and speedups are listed in Section D.9.

D TEXT PROMPTS FOR QUALITATIVE COMPARISON

We list the prompts used in Figs. 1 to 4 and 6 to 10, grouped by figure and given in their corresponding order.

D.1 PROMPTS FOR THE MAIN QUALITATIVE COMPARISON

The following twelve prompts correspond to Fig. 1.

Prompt①: Editorial photoshoot of an old woman, high fashion 2000s fashion.

Latency: 13.931 s; Speedup: $1.50\times$.

Prompt②: A man in a clown mask eating a donut.

Latency: 14.022 s; Speedup: $1.49\times$.

Prompt③: A cat wearing a halloween costume.

Latency: 13.925 s; Speedup: $1.51\times$.

Prompt④: Astronaut fox wearing space suit, detailed portrait, cinematic lighting, sci-fi style.

Latency: 13.960 s; Speedup: $1.50\times$.

Prompt⑤: CG art of a majestic castle, evoking a sense of splendor.

Latency: 14.791 s; Speedup: $1.42\times$.

Prompt⑥: Breathtaking view of the Colosseum against a sunny sky in Rome.

Latency: 14.335 s; Speedup: $1.46\times$.

Prompt⑦: The style of the image is a real photo depicting a single, fluffy sheep standing in a lush, green pasture. The sheep is prominent in the foreground, with its thick, curly white wool contrasting vividly against the vibrant green grass. Its face is slightly turned towards the viewer, displaying a gentle expression with soft, dark eyes and a small, damp nose. The background features a serene rural landscape, with rolling hills dotted with similar grazing sheep and a few sparse trees. The sky above is a clear blue, dotted sparsely with faint, wispy clouds. The light appears to be that of early morning, casting soft shadows and highlighting the texture of the sheep's wool and the fresh blades of grass. The color palette consists of calming natural tones, with the white of the wool standing out as a bright focal point. Overall, the image gives a detailed and clear representation of a pastoral scene centered on the tranquil presence of the sheep.

Latency (H200): 6.379 s; Speedup: $1.45\times$.

Prompt⑧: The style of the image is a realistic photo depicting a tall giraffe standing in a grassy savannah during the day. This giraffe, characterized by its uniquely patterned, patchy brown and white fur, stretches gracefully towards the sky. Its long neck, a distinguishing fea-

ture, is held high, and its small head with large eyes and tufted horns looks forward attentively. The giraffe's long legs, nearly equal in length, indicate its towering height, which allows it to reach high branches for foliage, its primary diet. Behind the giraffe, the scene is open and airy, dotted with sparse, flat-topped acacia trees typical of the savannah landscape. The background sky is a clear blue, suggesting a dry, sunny day. Below the giraffe's feet, the ground is covered with short, dry grass and occasional patches of earth, reflecting the arid environment. The light from the sun enhances the textures and colors of the scene, providing a vivid and sharp quality to the details, such as the fur's pattern and the rough tree bark.

Latency (H200): 6.250 s; Speedup: $1.48\times$.

Prompt⑨: The style of the image is real photo. In the center foreground, a vivid red umbrella with a smooth, shiny surface and a metallic handle is open, providing a stark contrast to the vibrant blue couch behind it. The couch, featuring a plush and velvety fabric, appears inviting with its overstuffed cushions and curved armrests. It sits against a pale, neutral-colored wall that subtly highlights both the umbrella and couch. On the wall above the couch, small framed pictures add a personal touch to the decor. The floor is covered with a grey, textured rug that complements the overall serene setting. Light streams in from a window out of view, casting soft shadows and enhancing the bright, saturated colors of both the umbrella and couch. This detailed arrangement in the room creates a harmonious balance between vivid colors and cozy domestic elements.

Latency (H200): 6.208 s; Speedup: $1.49\times$.

Prompt⑩: The style of the image is a real photo. A large, vibrant potted plant dominates the foreground, its lush green leaves sprawling outward, some with pointed tips, others rounded, suggesting a mix of species such as ferns and perhaps a small palm. The pot is simple, cylindrical, and terracotta-colored, enhancing the natural aesthetic. Beside the plant sits a backpack, partially obscured by the foliage. The backpack appears sturdy, crafted from a thick, durable fabric in a deep navy blue, with visible pockets and zippers, designed for functionality. It rests against a light-colored, textured wall, possibly in an indoor setting that receives ample natural light, evident from the bright but diffused lighting casting soft shadows on the ground. The ground is a hardwood floor, polished, with natural wood tones that complement the earthy colors of the potted plant and the wall.

Latency (H200): 6.293 s; Speedup: $1.47\times$.

Prompt⑪: The style of the image is a real photo. This picture displays a serene park setting featuring an empty wooden bench with a distinct grainy texture, set on a pathway made of smooth, gray paving stones. Next to the bench, there is a large ceramic vase, about half the height of the bench, with a glossy blue and white pattern that includes intricate floral designs. The vase holds a lush arrangement of various flowers, predominantly in shades of purple and yellow. Behind the bench and the vase, the background reveals a gently blurred garden, with hints of other green plants and possibly distant trees, suggesting a maintained but natural environment. The lighting in the photo suggests it's either early morning or late afternoon, as evidenced by the soft, warm glow that enhances the tranquility of the scene.

Latency (H200): 6.336 s; Speedup: $1.46\times$.

Prompt⑫: The style of the image is a real photo. The scene focuses on a single bicycle standing on a paved street, likely taken during the day given the bright natural lighting. This bicycle features a classic design with a slender steel frame painted in a shiny red, complementing the black rubber tires that appear well-inflated and ready for a ride. Mounted on the bicycle frame is a small, woven wicker basket, hinting at functionality for casual errands or leisurely picnics. The handlebars, curved gently upwards, are equipped with black grips and a small bell on the right side. The seat, upholstered in brown leather, matches the vintage aesthetic and looks moderately worn, suggesting frequent use. In the background, a blurred cityscape provides a context of urban environment, with partial views of store fronts and pedestrians, which adds a layer of everyday life bustling around the stationary bike. The pavement under

the bike is speckled with small shadows from what appears to be an overhead tree, adding a dynamic texture to the scene.

Latency (H200): 6.250 s; Speedup: 1.48×.

D.2 Prompt for GRN Refinement Dynamics

The following prompt corresponds to Fig. 2.

Prompt: A cat chasing butterflies in a garden.

D.3 Prompt for Noise-Based Refinement Analysis

The following prompt corresponds to Fig. 3.

Prompt: A photorealistic orange tabby cat sitting in a sunlit garden, detailed fur and green plants in the background.

D.4 Prompt for Query and Key/Value Sparsity Analysis

The following prompt corresponds to Fig. 4.

Prompt: A hummingbird drinking from a glass of lemonade.

D.5 Prompts for the Detailed Qualitative Comparison

The following four prompts correspond to Fig. 7.

Prompt①: A vibrant parrot holds a paintbrush-like grip with its beak, carefully painting colorful strokes on a small canvas in an artistic studio setting. Sunlight illuminates its brilliant feathers, with paint jars, scattered brushes, and vivid splashes of color surrounding the miniature easel.

Latency: 14.027 s; Speedup: 1.49×.

Prompt②: A confident cat rides a skateboard downhill, fur flowing in the wind as it zooms along a paved path. Its paws grip the board as autumn leaves scatter behind it, with sunlit neighborhood houses blurred by the sense of speed.

Latency: 14.429 s; Speedup: 1.45×.

Prompt③: A cheeseburger with juicy beef patties and melted cheese sits on top of a toilet shaped like a throne in the middle of a royal chamber. Velvet curtains, gilded pillars, and dramatic chandelier light emphasize the absurdly luxurious presentation.

Latency: 14.021 s; Speedup: 1.49×.

Prompt④: Create a mesmerizing image of three intricately designed potions displayed on an ornate antique wooden table within a charming old apothecary, featuring cobalt blue, crimson red, and emerald green potions in uniquely shaped glass bottles with detailed labels, magical symbols, and a warm stained-glass-lit atmosphere.

Latency: 14.590 s; Speedup: 1.44×.

D.6 Prompts for Active-Token Visualization

The following two prompts correspond to the left panel of Fig. 6.

Prompt①: A flamingo balancing on one leg while sipping tea.

Prompt②: A mountain peak covered in mist.

D.7 Prompts for Few-Step Visualization

The following two prompts correspond to Fig. 8.

Prompt①: A wizard resting near a quiet library, volumetric lighting.

Prompt②: A magnificent skyscraper featuring modern minimalist design, cinematic view.

D.8 Prompts for Additional Refinement Trajectories

The following six prompts correspond to Fig. 9.

Prompt①: A penguin sliding down an icy slope into the ocean.

Prompt②: A snail carrying a tiny house on its shell.

Prompt③: A squirrel holding an umbrella made of a leaf.

Prompt④: A raccoon washing an apple in a stream.

Prompt⑤: A waterfall cascading into a blue pool.

Prompt⑥: A winding road through a green valley.

D.9 Prompts and Runtimes for Additional Qualitative Comparisons

The following 24 prompts correspond to Fig. 10, ordered from left to right within each row and from top to bottom across rows. Prompts 1–6, 7–12, 13–18, and 19–24 correspond to the four rows, respectively.

Prompt①: Close up of an attractive bride, happy and smiling, looking up in the distance, vibrant neon lighting, background out of focus.

Latency: 13.973 s; Speedup: 1.50×.

Prompt②: Stylish woman posing confidently with oversized sunglasses.

Latency: 13.930 s; Speedup: 1.50×.

Prompt③: Cute boy, hair looking up to the stars, snow, beautiful lighting, painting style by Abe Toshiyuki.

Latency: 14.054 s; Speedup: 1.49×.

Prompt④: Beautiful lady, freckles, big smile, blue eyes, short ginger hair.

Latency: 13.980 s; Speedup: 1.50×.

Prompt⑤: A drawing of a young man with blue eyes.

Latency: 13.933 s; Speedup: 1.50×.

Prompt⑥: A man holding up a yellow frisbee while smiling.

Latency: 13.880 s; Speedup: $1.51\times$.

Prompt⑦: A jackrabbit with a green lightsaber, intense expression, jedi.

Latency: 13.999 s; Speedup: $1.50\times$.

Prompt⑧: An alpaca made of colorful building blocks, cyberpunk.

Latency: 13.995 s; Speedup: $1.50\times$.

Prompt⑨: A robot penguin wearing a top hat and playing a vintage trumpet under a rainbow.

Latency: 14.056 s; Speedup: $1.49\times$.

Prompt⑩: A photo of a red panda playing in a kindergarten.

Latency: 14.312 s; Speedup: $1.46\times$.

Prompt⑪: Chocolate eagle dessert sculpture on plate, realistic food art, pastry decoration.

Latency: 13.918 s; Speedup: $1.51\times$.

Prompt⑫: Dreamy puppy surrounded by floating bubbles.

Latency: 14.275 s; Speedup: $1.47\times$.

Prompt⑬: A lighthouse in a giant wave, origami style.

Latency: 14.004 s; Speedup: $1.50\times$.

Prompt⑭: Rectangular perfume on a camarin, white lights, bright, well lit environment.

Latency: 14.031 s; Speedup: $1.49\times$.

Prompt⑮: Fantasy sandcastle made of food, surrounded by ocean, intricate details, whimsical style.

Latency: 14.767 s; Speedup: $1.42\times$.

Prompt⑯: A hot air balloon floating above the clouds.

Latency: 14.573 s; Speedup: $1.44\times$.

Prompt⑰: A blue scooter is parked near a curb in front of a green vintage car.

Latency: 13.910 s; Speedup: $1.51\times$.

Prompt⑱: Vintage cars parked in garage or street, cinematic lighting, nostalgic atmosphere.

Latency: 14.334 s; Speedup: $1.46\times$.

Prompt⑲: Product photography of a perfume placed on a white marble table with pineapple coconut and lime next to it as decoration, white curtains, intricate details, realistic minimalist style, and a bright concise atmosphere.

Latency: 14.042 s; Speedup: $1.49\times$.

Prompt⑳: A quiet library glows with warm yellow light between tall shelves. Leather-bound books, polished wooden tables, and shaded reading lamps fill the room, while a comfortable armchair and faint dust motes create an inviting atmosphere of stillness and focused study.

Latency: 14.253 s; Speedup: $1.47\times$.

Prompt(21): A parrot stands in front of a mirror, animatedly talking to its own reflection in a bright room. Its vivid feathers and tilted head are sharply detailed, while sunlight on the mirror frame and nearby houseplants adds warmth to the amusing scene.

Latency: 14.001 s; Speedup: $1.50\times$.

Prompt(22): A colorful chameleon blends seamlessly into vibrant leaves, its skin matching the surrounding colors perfectly. Dew beads along the leaf veins, while filtered rainforest light reveals the fine texture of its scales and its curled tail gripping a slender branch.

Latency: 14.359 s; Speedup: $1.46\times$.

Prompt(23): A strong kangaroo stands in a modern gym, energetically boxing a hanging punching bag under bright indoor lights. Its muscular tail steadies its stance, while leather gloves, polished flooring, and blurred exercise equipment give the scene a lively athletic atmosphere.

Latency: 13.997 s; Speedup: $1.50\times$.

Prompt(24): A graceful deer stands at the edge of a frozen lake, gazing at its clear reflection beneath a pale winter sky. Frost coats the surrounding reeds, and soft winter light catches its delicate antlers above the smooth, glassy expanse of ice.

Latency: 14.219 s; Speedup: $1.47\times$.

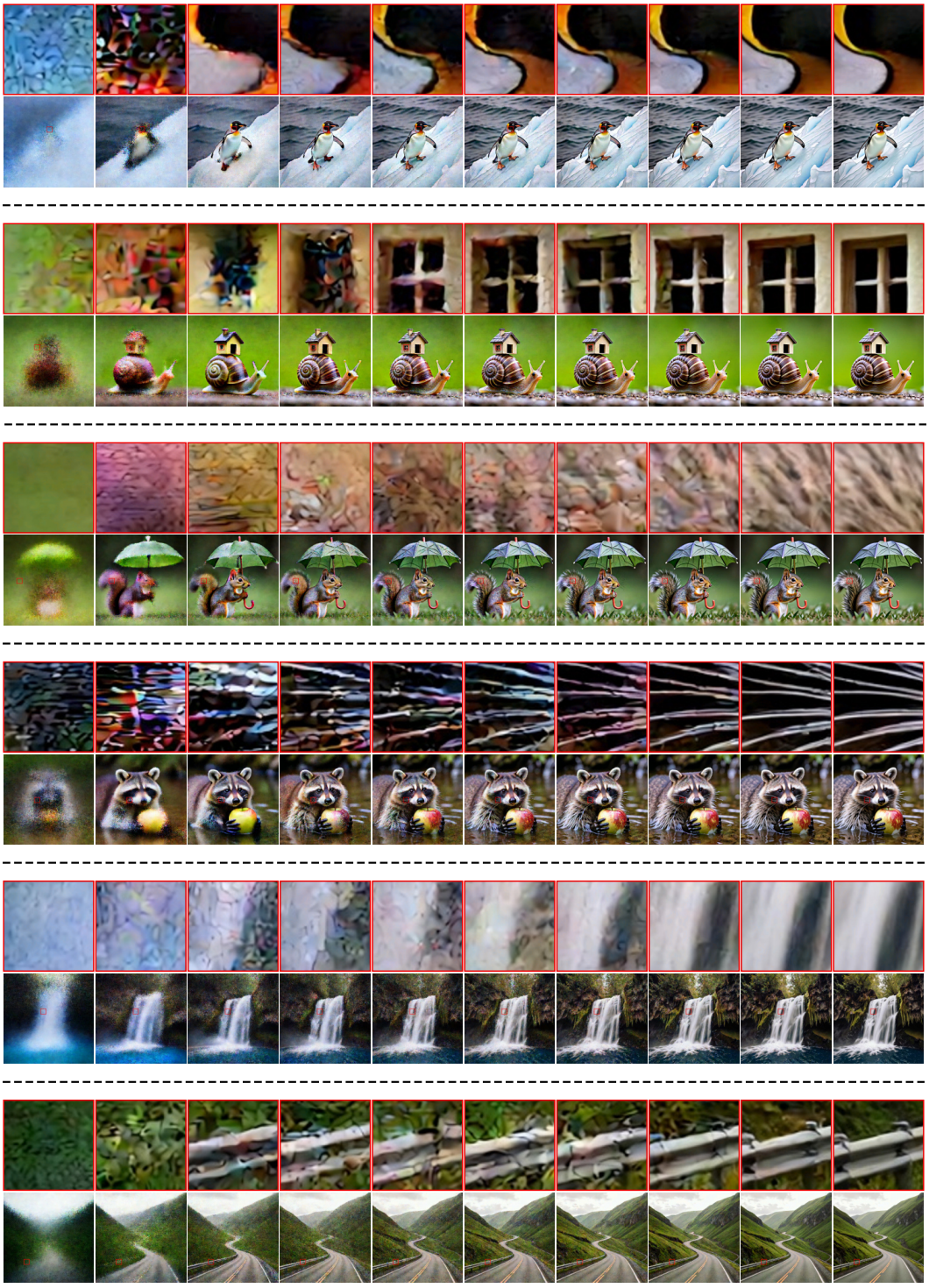


Figure 9: Additional GRN refinement trajectories. Each example shows full images below enlarged local regions. Columns progress from step 5 to step 50 in increments of five.

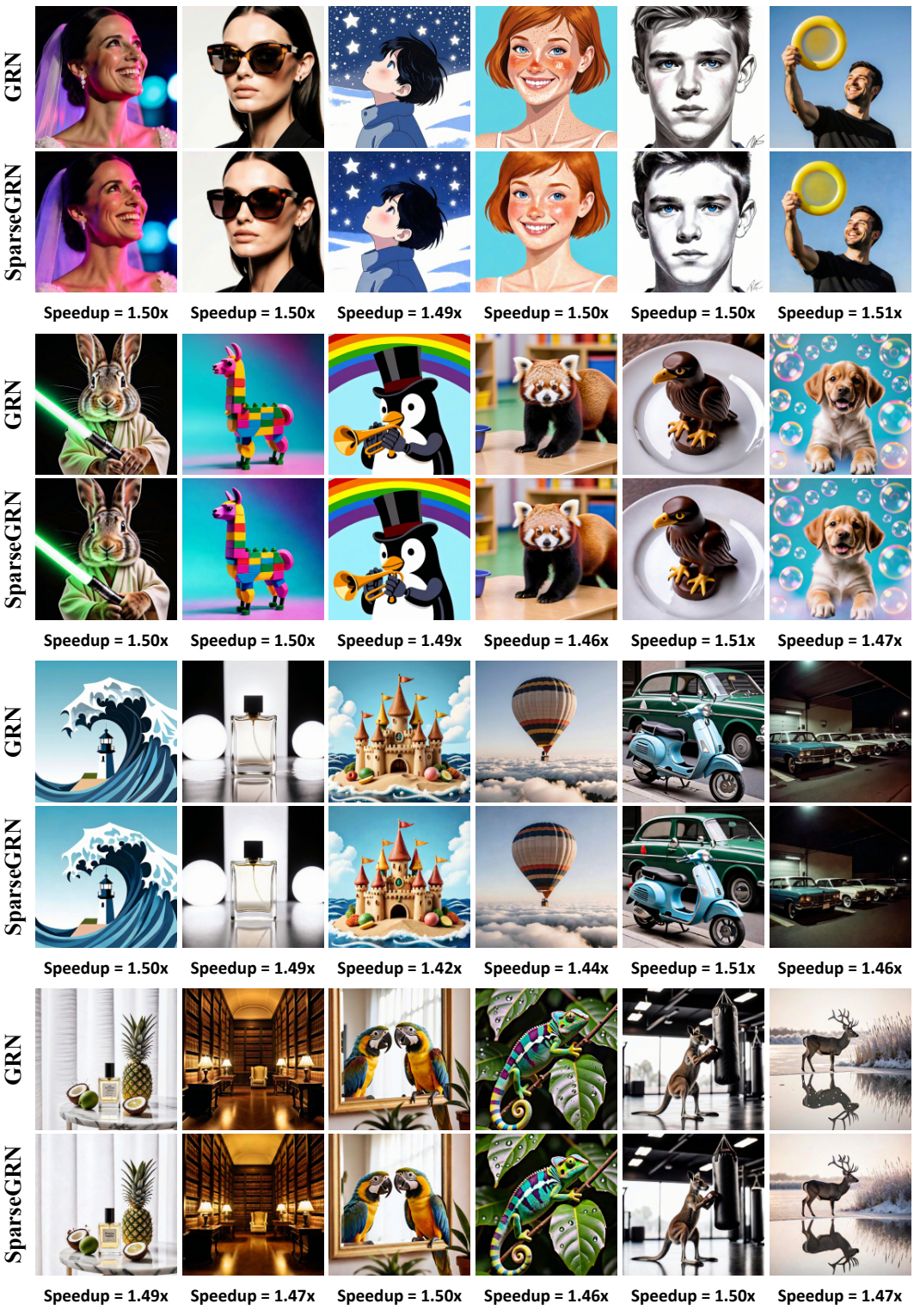


Figure 10: Additional qualitative comparison of GRN and ***SparseGRN***. The four groups show people, animals, scenes and objects, and long-prompt results. Each pair shows dense GRN above ***SparseGRN*** for the same prompt. Per-example speedups are annotated below the pairs.