SPECKCAST: SPECTRAL KALMAN FORECASTING FOR DIFFUSION ACCELERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion Transformers (DiTs) achieve high-fidelity image and video generation, but they suffer from substantial inference costs due to repeated transformer evaluations across many denoising steps. Feature caching reduces this cost by reusing features across denoising steps, and recent forecasting methods extend it by extrapolating features to skipped steps. However, these methods forecast the signal uniformly, even though its spatial-frequency components evolve at different stages of denoising. They also overlook that denoising progresses nonuniformly, so equal step intervals can correspond to very different amounts of change. These mismatches amplify forecast errors as more steps are skipped, limiting the achievable acceleration. We propose SpecKCast, a training-free method that forecasts the model output in the frequency domain. SpecKCast applies a 2D discrete cosine transform (DCT) to the output and tracks each coefficient with a lightweight Kalman filter along a band-specific time coordinate derived from the noise schedule. The filter maintains each coefficient's value and rate of change, and its forecast uncertainty determines when to evaluate the transformer. Experiments on FLUX.1-dev, Qwen-Image, Wan2.1-14B, and HunyuanVideo show that SpecKCast achieves a better trade-off between acceleration and fidelity than existing caching and forecasting methods, reaching up to $6\times$ speedup with negligible memory overhead.

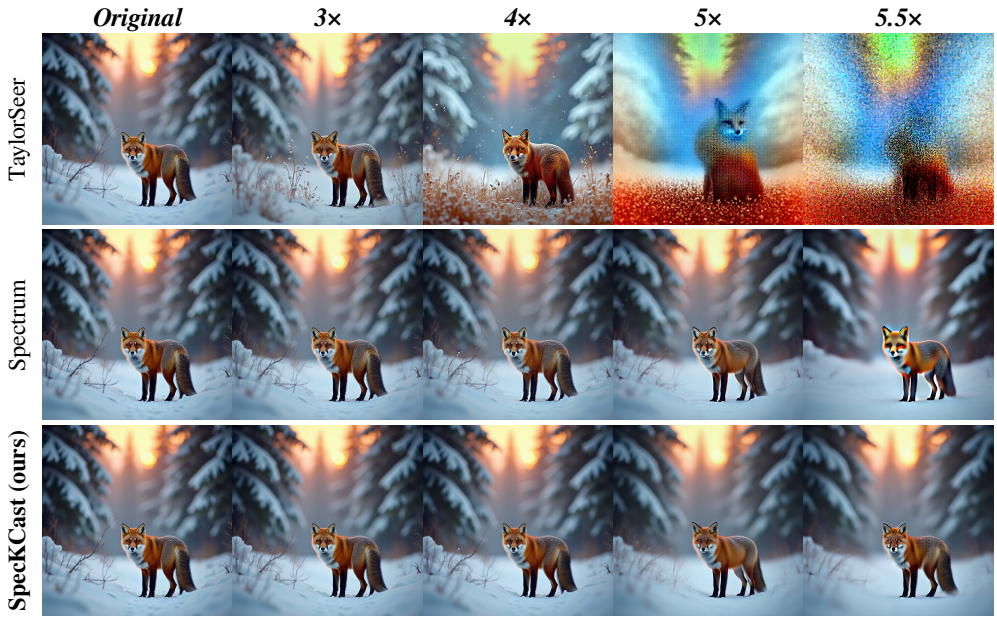

Figure 1: **Qualitative comparison** on FLUX at different acceleration ratios.

1 INTRODUCTION

Diffusion models (Ho et al., 2020) have become the dominant paradigm for high-fidelity visual generation, gaining significant attention from recent advances in image and video generation. Diffusion Transformers (DiTs) (Peebles & Xie, 2023) further improved generation quality by enabling

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.

greater model capacity. However, their costly forward passes throughout many denoising steps lead to substantially higher inference latency, limiting their use in real-time applications. To address this bottleneck, a wide range of acceleration techniques have been proposed to reduce either the step count or the per-step cost, including efficient attention (Xia et al., 2025), step distillation (Salimans & Ho, 2022), and quantization (Li et al., 2023).

Among these, feature caching has emerged as a practical, training-free approach. Motivated by the observation that model outputs are highly similar across adjacent timesteps, these methods cache features from previous steps and reuse them in subsequent ones, skipping substantial computation (Ma et al., 2024; Selvaraju et al., 2024). However, this similarity decays as the gap between timesteps grows, so naive reuse over longer spans significantly degrades generation quality, limiting caching to modest acceleration ratios. To address this, recent work shifts from reusing features to forecasting them (Liu et al., 2025b; Han et al., 2026). Rather than assuming that a cached feature remains valid at a later timestep, forecasting methods exploit temporal continuity along the denoising trajectory, using features from previous steps to estimate their evolution and extrapolate their values at the target timestep. By modeling temporal drift, forecasting remains effective across wider timestep gaps, enabling higher acceleration ratios at comparable generation quality.

Despite their promise, existing forecasting approaches impose uniformity on both what they forecast and how they model the signal over time (Liu et al., 2025b; Han et al., 2026). First, they extrapolate the model output or feature representation as a whole, or apply the same forecasting rule uniformly to all of its components (e.g., channels), even though the spatial-frequency components of the signal evolve very differently. As Figure 2 shows, low-frequency content changes sharply in the first few steps and then settles, while high-frequency content changes slightly at first but evolves rapidly toward the end. Because these bands are active at different stages, a single uniform forecasting may not serve them all. Second, existing methods use the sampler's step index or raw diffusion timestep as their temporal coordinate, even though the amount of denoising performed at each step is nonuniform and depends on the scheduler. Therefore, equal intervals in the chosen coordinate can represent very different amounts of denoising progress. Treating these intervals as equivalent makes the resulting trajectory harder to model and forecast. Together, these limitations restrict achievable acceleration, amplify forecast errors, and degrade generation quality as the skip length increases.

Motivated by this insight, we propose SpecKCast, a training-free method that rethinks both what we forecast and how we extrapolate. The key idea is to forecast the model's temporal evolution separately in the frequency domain, rather than uniformly in feature or latent space. SpecKCast transforms the model output into the spatial-frequency domain with a 2D discrete cosine transform (DCT), and forecasts each coefficient during denoising. Moreover, instead of indexing the trajectory by the sampler's timestep, SpecKCast exploits a time coordinate derived from the noise scheduler and applies a nonlinear time warping that improves the accuracy of local extrapolation within each band. To track each coefficient as it moves along its trajectory, SpecKCast uses a lightweight Kalman filter, a recursive state-space estimator that maintains the coefficient's value and rate of change, projecting this state across a skip and correcting it after a full denoising step. The filter also estimates the uncertainty of each forecast. As more steps are skipped, this uncertainty increases, while the rate of change decays to keep the prediction stable. Furthermore, SpecKCast partitions the coefficients into frequency bands that share a common temporal coordinate and forecast uncertainty, while each coefficient keeps its own value and rate of change. Finally, at skipped steps, the predicted coefficients are transformed back to the spatial domain and passed directly to the sampler.

The main contributions of this work are summarized as follows.

- **Frequency-Decoupled and Scheduler-Aligned Forecasting.** We introduce SpecKCast, a training-free method that decouples model output in the spatial-frequency domain and forecasts them along a nonlinear time coordinate derived from the noise scheduler.
- **Uncertainty-Aware Kalman Forecasting.** SpecKCast uses a lightweight Kalman filter to track frequency coefficient trajectories and leverage forecast uncertainty to maintain stability over skipped denoising steps.
- **State-of-the-Art Performance.** SpecKCast consistently improves the acceleration and generation quality trade-off over existing caching and forecasting methods while introducing negligible computational and memory overhead, requiring less than 5 MiB of additional memory for FLUX.1 (Labs, 2024).

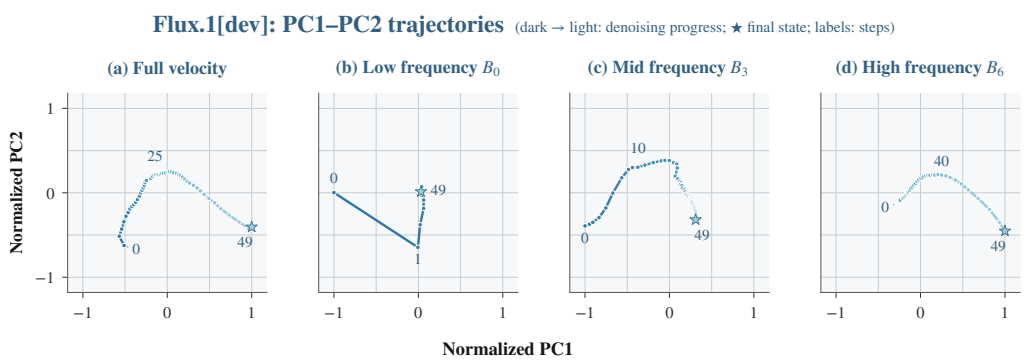

Figure 2: **Spatial-frequency bands of the denoising velocity follow different trajectories.** PCA trajectories of the 50 velocity predictions from a single FLUX.1-dev generation, projected onto the first two principal components. Larger gaps between consecutive points indicate larger changes. **(a)** Sampled entries of the full velocity. **(b–d)** Radial DCT bands: B_0 moves most in the first step, B_3 changes over a broader interval, and B_6 moves most after step 40. Each panel has its own PCA basis and normalization. Additional views, prompts, and Qwen-Image results are in Section E.

2 Related work

Diffusion models (Ho et al., 2020) have become a leading approach for image and video generation. Since the introduction of the Diffusion Transformer (DiT) (Peebles & Xie, 2023), many works have built on this architecture, achieving strong results across diverse domains (Chen et al., 2024a; Yang et al., 2025; Huang et al., 2025; Rajabi et al., 2026; Zhang et al., 2026). This progress, however, comes at a cost. Sampling requires many sequential forward passes through a large network, making inference slow and expensive. This limits practical applications and has made diffusion model acceleration an important research direction. A large body of work has therefore focused on reducing sampling steps, mostly by distilling a multi-step teacher into a few-step student (Meng et al., 2023). Examples include consistency models (Song & Dhariwal, 2024; Song et al., 2023) and DMD (Yin et al., 2024a;b). Although effective, these approaches require additional training and are prone to reduced sample diversity.

A more recent line of work explores feature caching, which accelerates inference without any additional training. The core idea is to reuse intermediate features across timesteps to avoid redundant computation (Zou et al., 2024; Gao et al., 2026; Xu et al., 2026; Peng et al., 2026). Early methods such as DeepCache (Ma et al., 2024) accelerated the U-Net backbone using handcrafted caching rules, and Δ-DiT (Chen et al., 2024b) later extended this idea to DiT architectures. Subsequent methods further refine what and when to cache. FORA (Selvaraju et al., 2024) reuses attention and MLP outputs across timesteps, while TeaCache (Liu et al., 2025a) exploits the correlation between timestep embeddings and model outputs to estimate how much the output changes between steps and decide when to reuse cached features. HiCache (Feng et al., 2026) forecasts features with scaled Hermite polynomials to capture their non-monotonic dynamics in DiTs. SeaCache (Chung et al., 2026) instead improves the caching decision by measuring feature changes after a spectral filter that preserves content-relevant components and suppresses noise. However, feature similarity decreases as the timestep gap increases. As more steps are skipped, the cached features deviate further from the actual ones, and these errors accumulate during sampling. This causes the trajectory to drift and degrades generation quality, limiting the achievable speedup.

To address this, recent work shifts from reusing features to forecasting them (Liu et al., 2025b; Han et al., 2026). Instead of assuming that cached features remain valid at later timesteps, forecasting methods use features from previous steps to predict how they evolve and estimate their values at the target timestep. TaylorSeer (Liu et al., 2025b) does this locally with a Taylor expansion, using finite differences of the most recent cached features, while Spectrum (Han et al., 2026) fits Chebyshev polynomials over all cached steps to better capture long-range dynamics. By modeling how features evolve, these methods remain effective over wider timestep gaps, enabling higher acceleration at comparable quality. However, both forecast uniformly in feature space and along the sampler's timestep, ignoring that denoising progress is nonuniform across steps. In contrast, SpecKCast forecasts each frequency band separately along a scheduler-derived time axis, using a lightweight Kalman filter that adapts its forecasts as the trajectory evolves.

3 BACKGROUND

3.1 DIFFUSION MODELS AND FLOW MATCHING

Diffusion models (Ho et al., 2020) define a generative process that gradually transforms samples from a simple prior (e.g., Gaussian) into data via a learned reverse-time process. Flow matching (Lipman et al., 2022; Liu et al., 2022) instead learns a velocity field that transports noise to data along near-straight paths. Both are instances of stochastic interpolants (Albergo et al., 2025), with $x_t = \alpha_t x_0 + \sigma_t \epsilon$, where rectified flow sets $\alpha_t = 1 - t$ and $\sigma_t = t$ and learns the velocity $v = \epsilon - x_0$. Sampling integrates the reverse flow over decreasing noise levels using Euler updates:

$$x_{j+1} = x_j + (\sigma_{j+1} - \sigma_j) Y_j, \qquad Y_j = v_\theta(x_j, \sigma_j, c), \tag{1}$$

where c denotes the text condition. We measure denoising cost by the number of function evaluations (NFE), defined as the total number of transformer forward passes.

3.2 FORECASTING FOR DIFFUSION ACCELERATION

Adjacent denoising steps produce highly correlated features and outputs, allowing redundant computation to be avoided (Ma et al., 2024; Selvaraju et al., 2024; Liu et al., 2025b; Han et al., 2026). Forecasting exploits this temporal redundancy by predicting skipped signals from past observations. Let $\mathcal{U} \subseteq \{0, \ldots, N-1\}$ denote the steps that evaluate the denoiser. For output-level forecasting,

$$\widehat{Y}_j = f(h_j, j), \qquad j \notin \mathcal{U}, \tag{2}$$

where h_j encodes past outputs and $\widehat{Y}_j$ replaces Y_j in the sampler. Forecasting may instead target intermediate features, making the choice of forecast target and representation a key design decision.

3.3 KALMAN FILTER

The Kalman filter (Kalman, 1960) is a classical method that recursively estimates the state of a dynamic signal from intermittent observations. It models the state transition as

$$h_j = A_j h_{j-1} + w_j, \qquad w_j \sim \mathcal{N}(0, Q_j), \tag{3}$$

$$y_j = H_j h_j + \eta_j, \qquad \eta_j \sim \mathcal{N}(0, R_j), \tag{4}$$

where A_j is the state-transition matrix, H_j maps the state to the observation, and Q_j and R_j are the process- and observation-noise covariances, respectively. The filter alternates between *prediction* and *update*, with updates only when observations are available. Given the previous estimate $\widehat{h}_{j-1}$ and covariance P_{j-1}, it predicts

$$\overline{h}_j = A_j \widehat{h}_{j-1}, \qquad \overline{P}_j = A_j P_{j-1} A_j^\top + Q_j. \tag{5}$$

When an observation y_j is available, the prediction is corrected using

$$K_j = \overline{P}_j H_j^\top \left(H_j \overline{P}_j H_j^\top + R_j \right)^{-1}, \tag{6}$$

$$\widehat{h}_j = \overline{h}_j + K_j (y_j - H_j \overline{h}_j), \qquad P_j = (I - K_j H_j) \overline{P}_j. \tag{7}$$

Here, K_j controls the balance between the new observation and the model prediction. An important property of the Kalman filter is that it is fully online, requiring a constant number of operations and no window of past observations to store or refit, so a large number of signals can be tracked in parallel at negligible cost. Between observations, the filter advances the state, while the covariance grows to reflect increasing forecast uncertainty, making the filter suitable for forecasting over extended gaps between observations.

4 METHOD

***Spec**tral **K**alman Fore**Cast**ing* (SpecKCast) is an inference-time acceleration method for DiTs that reduces transformer evaluations during denoising. Unlike feature-caching methods that approximate an intermediate activation and still execute the remaining network layers, SpecKCast models the final velocity consumed by the sampler. SpecKCast applies a 2D DCT to the model output and forecasts each frequency coefficient individually, while partitioning coefficients into bands that share forecast uncertainty and a noise-schedule-derived time coordinate, warped nonlinearly to better align with each band's temporal evolution. SpecKCast uses a Kalman filter to track each DCT coefficient's value and temporal derivative, while sharing covariance statistics within each frequency band. The predicted coefficients are transformed back to the spatial domain and passed directly to the sampler.

4.1 Frequency-Aware Forecasting

4.1.1 Spatial-frequency Decomposition

Forecasting the entire transformer output with a single temporal model ignores an important property of diffusion predictions: different spatial frequencies exhibit different temporal dynamics, as shown in Figure 2. Low-frequency content settles early and drifts slowly, while high-frequency content remains active until the end of sampling. To account for these distinct dynamics, SpecKCast transforms each output into the spatial-frequency domain and forecasts its coefficients separately.

Given the denoiser output $Y_j \in \mathbb{R}^{C\times H\times W}$ at timestep j,we apply an orthonormal 2D DCT independently to each channel, $Z_j = \mathrm{DCT}_{2\mathrm{D}}(Y_j)$, and partition the frequency plane into L non-overlapping radial bands. Each frequency coefficient is treated as a separate temporal signal and forecast by its own predictor, while the band determines how each coefficient's dynamics are modeled. On steps where the transformer computes the full denoising step, the observed Z_j updates the predictors; on skipped steps, the forecasters extrapolate each coefficient's trajectory and together form $\widehat{Z}_j$. The forecast is then transformed back to the spatial domain, $\widehat{Y}_j = \mathrm{IDCT}_{2\mathrm{D}}(\widehat{Z}_j)$, and passed directly to the diffusion sampler.

4.1.2 Band-specific Denoising Time

Each coefficient in SpecKCast is forecast as a function of time, which raises the question of what to use as the time variable. A natural choice is the step index j, but it does not capture the nonuniform progression of the denoising process. As a result, equal increments of j can correspond to very different amounts of change, making the trajectory harder to forecast. We instead derive the time variable from the *noise schedule*, expressed in log signal-to-noise ratio $\lambda_j = 2\log\big((1-\sigma_j)/(\sigma_j)\big)$. However, individual frequency bands still change most during different phases of denoising. A band's trajectory therefore bends where it is most active, and stays flat where it is not, making it nonlinear in λ_j. What the forecaster needs is a time variable whose increments reflect how much each band is expected to evolve. We therefore warp λ_j separately for each band ℓ,

$$t_\ell(j) \;=\; \mathrm{sigmoid}\Big(\tfrac{\lambda_j+\log p_\ell}{T_w}\Big), \qquad s_\ell(j) \;=\; \mathrm{rescale}_{[-1,1]}\big(t_\ell(j)\big), \tag{8}$$

where p_ℓ is the calibrated signal power of band ℓ (Section B.4) and T_w controls the strength. The resulting coordinate advances more when the band is expected to evolve and less when it is stable. This transformation only changes the forecasting coordinate and leaves the diffusion process unchanged.

4.2 Spectral Kalman Forecasting

SpecKCast uses a Kalman filter to track each frequency coefficient with a stepwise discrete dynamics model. The state of coefficient i at step j is $h_{i,j} = (m_{i,j}, \nu_{i,j})$, where $m_{i,j}$ is its estimated value and $\nu_{i,j}$ is its temporal derivative. Each coefficient maintains its own state, while the time coordinate s_ℓ, covariance, and process-noise statistics are shared within each band. Over each adjacent time increment $\Delta_{\ell,j} = s_\ell(j) - s_\ell(j-1)$, the state evolves

$$m_{i,j} = m_{i,j-1} + \Delta_{\ell,j}\nu_{i,j-1}, \qquad \nu_{i,j} = \rho_{\ell,j}\nu_{i,j-1}, \qquad \rho_{\ell,j} = e^{-\Delta_{\ell,j}/\tau_\ell}, \tag{9}$$

where τ_ℓ controls how quickly the temporal derivative decays, and the recursive update is applied at every sampler step, including skipped ones. The value $m_{i,j}$ advances according to the temporal derivative $\nu_{i,j}$, which gradually decays. This damping reduces the influence of outdated temporal changes and limits overshooting over extended skips. To incorporate these dynamics into the Kalman filter, we define the transition matrix $A_{\ell,j}$ and process-noise covariance $Q_{\ell,j}$ as

$$A_{\ell,j} = \begin{bmatrix} 1 & \Delta_{\ell,j} \\ 0 & \rho_{\ell,j} \end{bmatrix}, \qquad Q_{\ell,j} = q_{\ell,j} \begin{bmatrix} \Delta_{\ell,j}^3/3 & \Delta_{\ell,j}^2/2 \\ \Delta_{\ell,j}^2/2 & \Delta_{\ell,j} \end{bmatrix}, \tag{10}$$

Here, $q_{\ell,j}$ is the process-noise intensity, calibrated from the temporal activity of band ℓ and adapted during inference using observed forecast errors. The construction of $A_{\ell,j}$ and $Q_{\ell,j}$ is detailed in Section B.6. The covariance shared by coefficients in band ℓ evolves according to

$$P_{\ell,j} = A_{\ell,j}P_{\ell,j-1}A_{\ell,j}^\top + Q_{\ell,j}. \tag{11}$$

Initially, we set the state as $m_{i,0} = Z_{i,0}$ and $\nu_{i,0} = 0$. On compute steps, the observed coefficient $Z_{i,j}$ corrects the predicted state through the Kalman update. On skipped steps, the predicted coefficient values $m_{i,j}$ form the spectral forecast $\widehat{Z}_j$, which is transformed back to the spatial domain and passed to the sampler. Additionally, a separate covariance tracker estimates how each band's uncertainty grows across skipped steps. A compute step is triggered when the combined uncertainty exceeds a threshold.

4.3 Uncertainty-Aware Scheduling

The remaining question is which steps to compute. A fixed schedule is the simplest choice, but it cannot respond to changes in forecasting difficulty across generations and denoising phases. SpecKCast instead schedules adaptively, computing a step when the uncertainty accumulated by the Kalman filter indicates that the forecast is becoming unreliable. This requires combining the per-band covariances into a single decision quantity. Coefficients within the same band share the covariance $P_{\ell,j}$ because they use the same time coordinate and process-noise model. This avoids maintaining a separate covariance for every coefficient and reduces uncertainty tracking to only L matrices. We therefore aggregate the per-band uncertainties into a scalar *risk*,

$$R(j) = \omega(j) \sum_{\ell=1}^{L} d_\ell P_{11}^{(\ell)}(j), \qquad \omega(j) = \exp\left(-\beta \frac{j}{N-1}\right), \tag{12}$$

where d_ℓ is the number of coefficients in band ℓ and β controls how strongly late-stage risk is attenuated. Here, $P_{11}^{(\ell)}(j)$ is the top-left entry of $P_{\ell,j}$. Since the rows and columns of $P_{\ell,j}$ follow the state ordering $h_{i,j} = (m_{i,j}, \nu_{i,j})$, this entry represents the predicted error variance of the coefficient value $m_{i,j}$. Multiplying it by d_ℓ approximates the total forecast uncertainty of band ℓ. Accordingly, scheduling follows a simple threshold rule. After each compute step, we apply the covariance update to predict the risk at future steps. SpecKCast uses forecasted outputs until the first step k satisfying $R(k) > \varepsilon$, at which point the transformer is evaluated. A smaller ε triggers more frequent evaluations, while a larger value permits longer skips. We calibrate ε for the desired compute budget, while the covariance adapts to the observed forecast errors during generation.

5 Experiments

Setup. We evaluate SpecKCast on two state-of-the-art text-to-image models, FLUX.1-dev (Labs, 2024) at 1024×1024 and Qwen-Image (Wu et al., 2025) at 1328×1328, and further on two text-to-video models, Wan2.1-14B (Wan et al., 2025) and HunyuanVideo (Kong et al., 2024). Following common practice, we use 50-step sampling as the full-computation reference for all models. We compare against FORA (Selvaraju et al., 2024), TeaCache (Liu et al., 2025a), HiCache (Feng et al., 2026), SeaCache (Chung et al., 2026), TaylorSeer (Liu et al., 2025b), and Spectrum (Han et al., 2026). Among them, TaylorSeer and Spectrum are the most closely related to our method, as they also forecast future features across timesteps instead of directly reusing cached ones. All baselines are evaluated using their official implementations. We decompose each prediction into $L = 7$ frequency bands and, unless otherwise stated, set $\tau = 0.1$, $T_w = 2.0$, and $\beta = 0.6$. All experiments are conducted on NVIDIA H100 GPUs.

Evaluation. For text-to-image generation, we use the DrawBench (Saharia et al., 2022) prompts and report ImageReward (Xu et al., 2023) and CLIP Score (Hessel et al., 2021), which measures text–image alignment. For text-to-video generation, we follow the standard prompts and protocol of VBench (Huang et al., 2024) and report its Quality Score, which aggregates dimensions such as temporal consistency, motion smoothness, and per-frame aesthetic and imaging quality. Since an accelerated sampler should reproduce the output of the full model rather than merely produce a plausible sample, we also compare each accelerated sample with its 50-step reference generated from the same prompt and seed. We report PSNR for pixel-level error, SSIM (Wang et al., 2004) for structural similarity, and LPIPS (Zhang et al., 2018) for perceptual distance. For efficiency, we report sampling latency, speedup over the 50-step reference, and memory overhead (Mem. Δ), defined as the peak allocated GPU memory above that of Base at the same NFE.

Table 1: Quantitative comparison in text-to-image generation on FLUX.1-dev and Qwen-Image.

		FLUX.1-dev (1024×1024)								**Qwen-Image** (1328×1328)							
		Acceleration		Mem. Δ	Quality			Image	CLIP↑	Acceleration		Mem. Δ	Quality			Image	CLIP↑
NFE	**Method**	Latency(s)↓	Speedup↑	(GiB)↓	PSNR↑	SSIM↑	LPIPS↓	Reward↑		Latency(s)↓	Speedup↑	(GiB)↓	PSNR↑	SSIM↑	LPIPS↓	Reward↑	
50	Reference	11.68	1.00	–	–	–	–	0.98	27.20	52.22	1.00	–	–	–	–	1.24	29.14
≈ 14	Base	3.35	3.48	0.00	15.30	0.633	0.423	0.98	27.51	14.83	3.52	0.00	15.18	0.640	0.393	1.08	29.19
	FORA	3.98	2.93	2.01	15.03	0.628	0.433	0.96	27.48	16.11	3.24	9.46	14.03	0.571	0.456	1.02	28.97
	TeaCache	3.55	3.29	0.07	16.86	0.684	0.355	**1.01**	**27.57**	15.68	3.33	0.20	19.42	0.781	0.229	1.19	**29.36**
	HiCache	4.42	2.64	4.02	18.02	0.708	0.292	1.00	27.50	19.16	2.72	18.97	16.04	0.647	0.335	1.16	29.08
	SeaCache	3.59	3.25	0.09	22.30	0.813	0.184	0.98	27.40	15.78	3.31	0.24	21.43	0.821	0.190	1.20	29.11
	TaylorSeer	4.59	2.55	4.02	17.74	0.700	0.305	1.00	27.32	18.37	2.84	18.97	14.22	0.598	0.423	1.05	28.58
	Spectrum	3.49	3.36	1.15	23.60	0.814	0.165	0.98	27.33	15.72	3.32	2.84	23.66	0.851	0.135	1.21	29.16
	SpecKCast	**3.45**	**3.38**	**0.005**	**24.28**	**0.836**	**0.146**	0.98	27.32	**15.01**	**3.48**	**0.01**	**24.16**	**0.852**	**0.134**	**1.22**	29.17
≈ 10	Base	2.48	4.71	0.00	14.55	0.599	0.483	0.87	27.31	10.60	4.93	0.00	14.03	0.586	0.480	0.95	28.89
	FORA	3.16	3.70	2.01	14.32	0.591	0.502	0.84	27.40	12.16	4.29	9.46	12.73	0.502	0.561	0.65	28.15
	TeaCache	2.56	4.57	0.07	15.66	0.631	0.440	0.97	27.50	11.46	4.56	0.20	16.11	0.656	0.390	1.06	28.84
	HiCache	3.62	3.23	4.02	16.30	0.635	0.390	0.99	27.60	15.41	3.39	18.97	12.77	0.488	0.519	0.88	28.02
	SeaCache	2.55	4.59	0.09	19.29	0.731	0.304	0.97	27.52	11.75	4.45	0.24	19.61	0.767	0.257	1.15	29.08
	TaylorSeer	3.55	3.29	4.02	15.28	0.607	0.436	0.97	27.37	14.40	3.63	18.97	10.79	0.405	0.623	0.62	26.85
	Spectrum	2.55	4.59	0.87	21.27	0.729	0.261	0.98	27.62	11.53	4.53	2.21	20.85	0.746	0.318	1.17	**29.80**
	SpecKCast	**2.46**	**4.74**	**0.005**	**21.77**	**0.749**	**0.258**	0.98	27.57	**10.80**	**4.83**	**0.01**	**22.28**	**0.790**	**0.201**	**1.19**	29.01
≈ 8	Base	1.98	5.89	0.00	14.29	0.578	0.518	0.81	27.39	8.54	6.11	0.00	13.53	0.561	0.531	0.80	28.78
	FORA	2.70	4.33	2.01	14.12	0.574	0.529	0.74	27.07	10.15	5.15	9.46	12.39	0.475	0.589	0.40	27.76
	TeaCache	2.13	5.75	0.07	15.07	0.600	0.506	0.82	27.48	11.50	4.54	0.20	16.00	0.648	0.397	1.04	28.83
	HiCache	3.23	3.48	4.02	15.13	0.582	0.466	**0.98**	27.53	13.50	3.87	18.97	10.86	0.391	0.639	0.56	27.21
	SeaCache	2.13	5.48	0.09	18.47	**0.688**	0.370	0.96	27.52	10.38	5.03	0.24	18.41	0.733	0.307	1.09	28.95
	TaylorSeer	3.14	3.72	4.02	13.80	0.543	0.516	0.93	27.16	12.48	4.18	18.97	9.17	0.322	0.720	0.26	25.90
	Spectrum	2.15	5.43	0.73	18.71	0.632	0.397	0.90	27.73	10.23	5.48	1.89	16.91	0.604	0.527	0.90	**29.68**
	SpecKCast	**2.11**	**5.54**	**0.005**	**19.81**	0.683	**0.353**	0.89	27.48	**8.70**	**6.00**	**0.01**	**20.44**	**0.740**	**0.267**	**1.11**	28.93

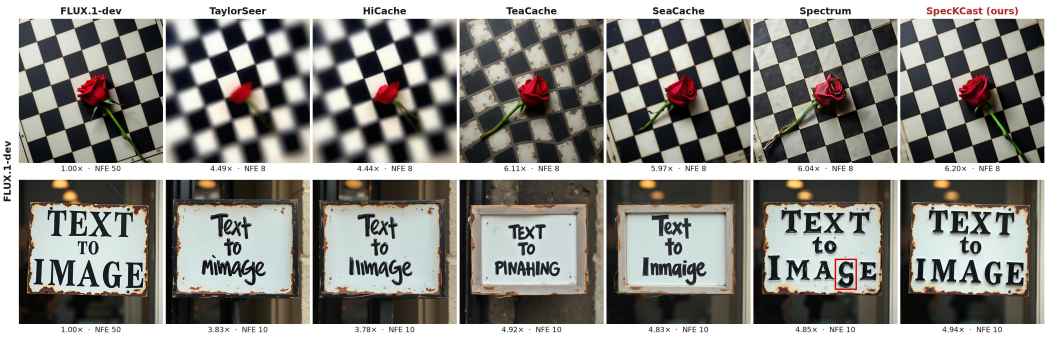

Figure 3: Qualitative comparison on text-to-image generation using FLUX.1-dev.

5.1 Text-to-image Generation

Table 1 reports results on FLUX.1-dev and Qwen-Image, where Base runs the original sampler with the same NFE and no caching. SpecKCast achieves the highest speedup among acceleration methods in every setting, along with the best PSNR and LPIPS in all settings and the best SSIM in all but one. At the highest acceleration, it outperforms the strongest baseline by 1.1 dB PSNR on FLUX.1-dev ($5.54\times$) and 2.0 dB on Qwen-Image ($6.00\times$). Spectrum, the closest competitor at 14 and 10 NFE, trails SpecKCast by 0.5–1.4 dB PSNR and degrades sharply on Qwen-Image at 8 NFE, where its LPIPS rises to 0.527 compared with 0.267 for SpecKCast. Compared with Base, which runs at a similar speed, SpecKCast improves PSNR by 5.5–9.0 dB, showing that it tracks the 50-step trajectory far more closely than simply taking fewer steps. These gains cost at most 0.01 GiB of memory, whereas TaylorSeer, HiCache, and Spectrum add 0.73–18.97 GiB. Figures 3 and 5 show qualitative results on FLUX.1-dev and Qwen-Image, respectively. TaylorSeer and HiCache blur the images, TeaCache and SeaCache alter textures and lighting, and every baseline misspells or distorts the text on the sign. SpecKCast stays closest to the 50-step reference while running fastest. More samples are provided in Section D.

5.2 Text-to-video Generation

Table 2 reports results on Wan2.1-14B and HunyuanVideo. SpecKCast achieves the highest or near-highest speedup in every setting, along with the best SSIM in all settings and the best PSNR and LPIPS in most. On HunyuanVideo at the 14-NFE budget, it improves PSNR by 2.1 dB over SeaCache, the strongest baseline, while running at $3.33\times$ instead of $2.57\times$. The advantage holds at the

Table 2: Quantitative comparison in text-to-video generation on Wan2.1-14B and HunyuanVideo.

		Wan2.1-14B							HunyuanVideo						
		Acceleration		Mem. Δ	Quality			VBench	Acceleration		Mem. Δ	Quality			VBench
NFE	Method	Latency(s)↓	Speedup↑	(GiB)↓	PSNR↑	SSIM↑	LPIPS↓	Quality↑	Latency(s)↓	Speedup↑	(GiB)↓	PSNR↑	SSIM↑	LPIPS↓	Quality↑
50	Reference	370	1.00	–	–	–	–	81.11	183	1.00	–	–	–	–	80.26
≈ 14	Base	116	3.20	0.00	13.95	0.471	0.518	79.81	58.3	3.14	0.00	17.33	0.641	0.417	78.38
	FORA	108.3	3.42	0.62	21.62	0.753	0.192	80.63	56.8	3.22	0.19	23.46	0.775	0.216	78.90
	TeaCache	**105.2**	**3.52**	0.94	22.13	0.772	0.191	80.24	61.2	2.99	0.37	20.56	0.709	0.279	79.35
	HiCache	111.9	3.31	3.75	18.42	0.675	0.267	80.12	56.8	3.22	1.12	23.43	0.785	0.198	79.54
	SeaCache	113.2	3.27	4.37	23.10	0.795	0.157	80.38	71.3	2.57	0.37	27.59	0.864	0.121	79.53
	TaylorSeer	109.4	3.38	4.69	17.94	0.667	0.264	80.12	57.2	3.20	1.49	22.29	0.767	0.217	**79.98**
	Spectrum	110.3	3.35	13.12	22.93	0.789	**0.136**	80.76	58.0	3.15	3.56	26.95	0.831	0.133	79.32
	SpecKCast	**105.2**	**3.52**	**0.05**	**24.47**	**0.813**	0.140	**80.86**	**55.0**	**3.33**	**0.03**	**29.72**	**0.884**	**0.107**	79.68
≈ 10	Base	80.50	4.59	0.00	13.66	0.452	0.578	78.05	42.7	4.28	0.00	16.73	0.621	0.469	76.59
	FORA	79.6	4.65	0.62	20.68	0.709	0.270	79.76	43.0	4.26	0.19	22.03	0.726	0.306	77.12
	TeaCache	80.3	4.61	0.94	21.11	0.732	0.245	79.05	47.2	3.88	0.37	19.65	0.672	0.329	78.19
	HiCache	80.3	4.61	3.75	14.04	0.519	0.474	74.96	43.0	4.26	1.12	21.17	0.718	0.293	78.06
	SeaCache	89.5	4.13	4.37	19.53	0.686	0.271	79.87	45.4	4.03	0.37	19.34	0.698	0.338	78.08
	TaylorSeer	80.2	4.61	4.69	14.57	0.549	0.404	76.20	43.4	4.22	1.49	19.54	0.689	0.318	78.19
	Spectrum	80.8	4.58	9.38	21.41	0.733	**0.175**	80.42	43.8	4.17	2.81	24.60	0.767	0.191	78.80
	SpecKCast	**79.1**	**4.68**	**0.05**	**21.62**	**0.750**	0.181	**80.48**	**42.7**	**4.29**	**0.03**	**25.62**	**0.821**	**0.172**	**79.25**
≈ 8	Base	80.2	4.61	0.00	13.56	0.449	0.614	76.98	38.2	4.79	0.00	16.35	0.612	0.510	75.01
	FORA	65.5	5.65	0.62	**20.19**	0.669	0.336	78.38	36.2	5.06	0.19	21.22	0.655	0.346	75.00
	TeaCache	65.8	5.62	0.94	19.61	0.656	0.335	76.52	36.9	4.96	0.37	18.37	0.617	0.415	76.00
	HiCache	66.3	5.58	3.75	12.06	0.430	0.578	72.01	36.2	5.06	1.12	19.83	0.643	0.346	75.42
	SeaCache	**65.0**	**5.69**	4.37	15.27	0.519	0.482	76.97	37.0	4.95	0.37	18.98	0.681	0.380	**76.81**
	TaylorSeer	65.7	5.63	4.69	13.02	0.480	0.475	73.57	36.5	5.01	1.49	18.15	0.620	0.387	74.90
	Spectrum	66.0	5.61	7.50	17.81	0.638	0.303	78.74	37.9	4.82	2.43	21.74	0.674	0.297	75.77
	SpecKCast	65.3	5.67	**0.05**	19.94	**0.683**	**0.271**	**79.20**	**35.8**	**5.12**	**0.03**	**22.34**	**0.747**	**0.295**	76.32

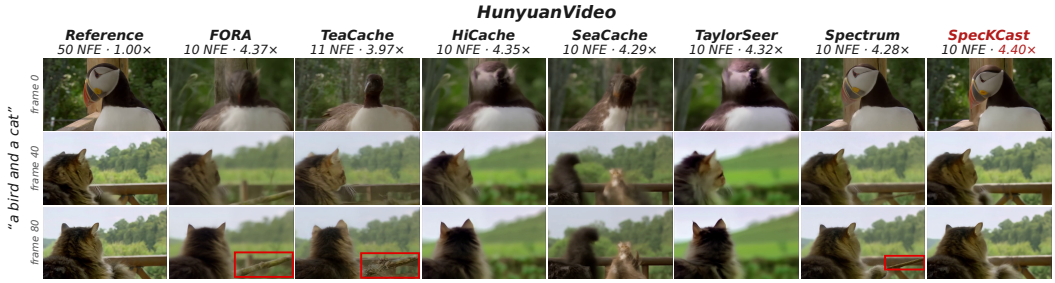

Figure 4: Qualitative comparison on text-to-video generation using HunyuanVideo.

8-NFE budget, where SpecKCast keeps the best SSIM and LPIPS on both models, while HiCache and TaylorSeer drop to 12.06 and 13.02 dB PSNR on Wan2.1-14B. SpecKCast also achieves the best VBench Quality on Wan2.1-14B at every budget, staying within 0.25 points of the reference at the 14-NFE budget. Spectrum obtains slightly lower LPIPS on Wan2.1-14B at the 14- and 10-NFE budgets, but requires up to 13.12 GiB of extra memory, compared with 0.05 GiB for SpecKCast. Figures 4 and 6 show qualitative results on HunyuanVideo and Wan2.1-14B, respectively. On HunyuanVideo, most baselines replace the puffin in the first frame with a blurred or different bird, and SeaCache adds a second cat in later frames. On Wan2.1-14B, the baselines introduce local artifacts (red boxes), mostly in the window background, and HiCache also shifts the lighting. SpecKCast stays closest to the 50-step reference across frames on both models.

5.3 Ablation Studies

Table 3 ablates the components of SpecKCast on FLUX.1-dev at 14 NFE. Since uncertainty-aware scheduling relies on the Kalman covariance, we first compare forecasters under a fixed schedule. Kalman and Taylor perform similarly, with Kalman slightly faster and higher in PSNR, while Chebyshev is both slower and less accurate. With our frequency decomposition and time warping, the Taylor forecaster also outperforms TaylorSeer (Table 1) by 4.2 dB PSNR while reducing latency from 4.59 to 3.46 s. Replacing the fixed schedule with uncertainty-aware scheduling (Section 4.3) improves PSNR by 2.2 dB. Band-specific time warping (Section 4.1.2) outperforms the raw log-SNR λ_j by 3.3 dB and the step index j by 4.7 dB, and removing the frequency decomposition lowers PSNR by 3.1 dB. Overall, the full model achieves the best PSNR, SSIM, LPIPS, and ImageReward. Since all variants except Chebyshev run within 0.1 s of each other, these gains come at almost no extra latency.

Table 3: Ablation of SpecKCast components on FLUX.1-dev at 14 NFE.

Forecaster	Scheduler	Time variable	Frequency decomposition	Latency(s)↓	PSNR↑	SSIM↑	LPIPS↓	Image Reward↑	CLIP↑
Taylor	fixed	warped-logSNR	✓	3.46	21.90	0.811	0.171	0.978	27.24
Chebyshev	fixed	warped-logSNR	✓	3.97	21.10	0.761	0.252	0.975	**27.66**
Kalman	fixed	warped-logSNR	✓	**3.43**	22.09	0.810	0.178	0.973	27.27
Kalman	adaptive	raw-logSNR	✓	3.52	21.03	0.775	0.210	0.977	27.24
Kalman	adaptive	uniform step index	✓	3.51	19.54	0.740	0.256	0.969	27.28
Kalman	adaptive	warped-logSNR	×	3.48	21.19	0.785	0.201	0.966	27.13
Kalman	adaptive	warped-logSNR	✓	3.45	**24.28**	**0.836**	**0.146**	**0.980**	27.32

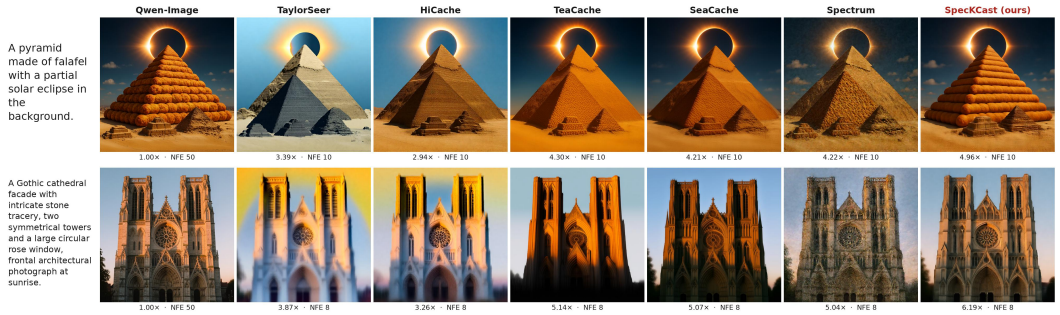

Figure 5: Qualitative comparison on text-to-image generation using Qwen-Image.

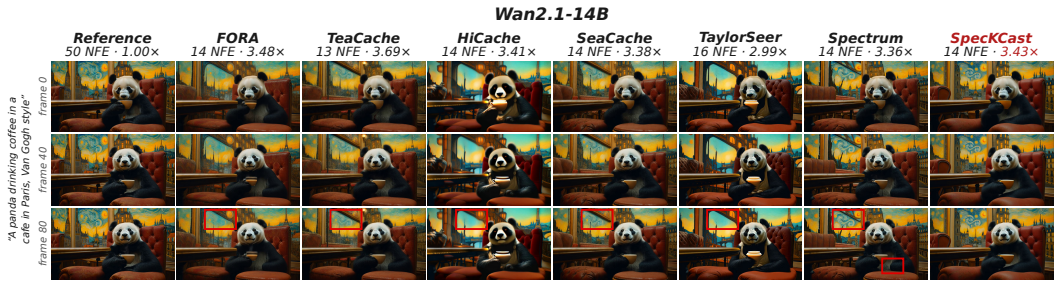

Figure 6: Qualitative comparison on text-to-video generation using Wan2.1-14B.

6 Conclusion

We presented SpecKCast, a training-free method for accelerating Diffusion Transformers by forecasting the denoiser output at skipped sampling steps. Rather than modeling the output uniformly, SpecKCast decomposes it into spatial-frequency components and forecasts their distinct temporal dynamics along band-specific, noise-schedule-aligned time coordinates. A lightweight Kalman filter tracks the value and rate of change of each DCT coefficient, while its uncertainty provides a principled signal for adaptively determining when a full transformer evaluation is needed. Across image and video diffusion models, SpecKCast consistently improves the acceleration–quality tradeoff over existing caching and forecasting methods, achieving the best fidelity to the 50-step reference in most settings with negligible latency and memory overhead. These results highlight the benefit of explicitly accounting for the distinct temporal evolution of frequency components when forecasting diffusion trajectories. SpecKCast, however, targets only temporal redundancy across denoising steps; each computed step still processes the full token grid and therefore retains the spatial redundancy within a transformer evaluation. Jointly exploiting temporal and spatial redundancy could enable further acceleration and represents a promising direction for future work.

AI USE STATEMENT

In this work, we used generative AI tools to edit the manuscript for grammar, clarity, readability, and presentation. We did not use generative AI tools for research ideation, developing theoretical models or hypotheses, formulating or proving mathematical claims, implementing methods, generating or processing data, translation, or analyzing and interpreting results. All AI-edited text was carefully reviewed and revised by the authors to ensure that it accurately preserves the intended technical meaning and remains consistent with the authors' original ideas and contributions. We take full responsibility for the final content of this work.

REFERENCES

Michael Albergo, Nicholas M Boffi, and Eric Vanden-Eijnden. Stochastic interpolants: A unifying framework for flows and diffusions. *Journal of Machine Learning Research*, 26(209):1–80, 2025.

Junsong Chen, Chongjian Ge, Enze Xie, Yue Wu, Lewei Yao, Xiaozhe Ren, Zhongdao Wang, Ping Luo, Huchuan Lu, and Zhenguo Li. Pixart-σ: Weak-to-strong training of diffusion transformer for 4k text-to-image generation. In *European Conference on Computer Vision*, pp. 74–91. Springer, 2024a.

Pengtao Chen, Mingzhu Shen, Peng Ye, Jianjian Cao, Chongjun Tu, Christos-Savvas Bouganis, Yiren Zhao, and Tao Chen. Δ-DiT: A training-free acceleration method tailored for diffusion transformers. *arXiv preprint arXiv:2406.01125*, 2024b.

Jiwoo Chung, Sangeek Hyun, MinKyu Lee, Byeongju Han, Geonho Cha, Dongyoon Wee, Youngjun Hong, and Jae-Pil Heo. Seacache: Spectral-evolution-aware cache for accelerating diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14283–14294, 2026.

Liang Feng, Shikang Zheng, Jiacheng Liu, Yuqi Lin, Qinming Zhou, Peiliang Cai, Xinyu Wang, Junjie Chen, Chang Zou, Yue Ma, et al. Hicache: A plug-in scaled-hermite upgrade for taylor-style cache-then-forecast diffusion acceleration. In *International Conference on Learning Representations*, volume 2026, pp. 112361–112386, 2026.

Huanlin Gao, Ping Chen, Fuyuan Shi, Ruijia Wu, Li YanTao, Qiang Hui, Yuren You, Ting Lu, Chao Tan, Shaoan Zhao, Zhaoxiang Liu, Fang Zhao, Kai Wang, and Shiguo Lian. Meancache: From instantaneous to average velocity for accelerating flow matching inference. In *International Conference on Learning Representations*, volume 2026, pp. 60466–60485, 2026.

Jiaqi Han, Juntong Shi, Puheng Li, Haotian Ye, Qiushan Guo, and Stefano Ermon. Adaptive spectral feature forecasting for diffusion sampling acceleration. *arXiv preprint arXiv:2603.01623*, 2026.

Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. Clipscore: A reference-free evaluation metric for image captioning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 7514–7528, 2021.

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

Longfei Huang, Yu Liang, Hao Zhang, Jinwei Chen, Wei Dong, Lunde Chen, Wanyu Liu, Bo Li, and Peng-Tao Jiang. Sdmatte: Grafting diffusion models for interactive matting. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15229–15239. IEEE, 2025.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, et al. Vbench: Comprehensive benchmark suite for video generative models. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21807–21818. IEEE, 2024.

R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960. doi: 10.1115/1.3662552.

Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, et al. Hunyuanvideo: A systematic framework for large video generative models. *arXiv preprint arXiv:2412.03603*, 2024.

Black Forest Labs. Flux. `https://github.com/black-forest-labs/flux`, 2024.

Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-diffusion: Quantizing diffusion models. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17489–17499. IEEE, 2023.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Feng Liu, Shiwei Zhang, Xiaofeng Wang, Yujie Wei, Haonan Qiu, Yuzhong Zhao, Yingya Zhang, Qixiang Ye, and Fang Wan. Timestep embedding tells: It's time to cache for video diffusion model. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7353–7363. IEEE, 2025a.

Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Junjie Chen, and Linfeng Zhang. From reusing to forecasting: Accelerating diffusion models with taylorseers. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15853–15863. IEEE, 2025b.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15762–15772. IEEE, 2024.

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

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182. IEEE, 2023.

Xurui Peng, Chenqian Yan, Hong Liu, Rui Ma, Fangmin Chen, Xing Wang, Zhihua Wu, Songwei Liu, and Mingbao Lin. Ertacache: Error rectification and timesteps adjustment for efficient diffusion. In *International Conference on Learning Representations*, volume 2026, pp. 82104–82120, 2026.

Javad Rajabi, Kimia Shaban, Koorosh Roohi, David B Lindell, and Babak Taati. Sega: Spectral-energy guided attention for resolution extrapolation in diffusion transformers. *arXiv preprint arXiv:2605.22668*, 2026.

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.

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

Pratheba Selvaraju, Tianyu Ding, Tianyi Chen, Ilya Zharkov, and Luming Liang. Fora: Fast-forward caching in diffusion transformer acceleration. *arXiv preprint arXiv:2407.01425*, 2024.

Yang Song and Prafulla Dhariwal. Improved techniques for training consistency models. In *International Conference on Learning Representations*, volume 2024, pp. 15078–15097, 2024.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. *arXiv preprint arXiv:2303.01469*, 2023.

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, Jianyuan Zeng, Jiayu Wang, Jingfeng Zhang, Jingren Zhou, Jinkai Wang, Jixuan Chen, Kai Zhu, Kang Zhao, Keyu Yan, Lianghua Huang, Mengyang Feng, Ningyi Zhang, Pandeng Li, Pingyu Wu, Ruihang Chu, Ruili Feng, Shiwei Zhang, Siyang Sun, Tao Fang, Tianxing Wang, Tianyi Gui, Tingyu Weng, Tong Shen, Wei Lin, Wei Wang, Wei Wang, Wenmeng Zhou, Wente Wang, Wenting Shen, Wenyuan Yu, Xianzhong Shi, Xiaoming Huang, Xin Xu, Yan Kou, Yangyu Lv, Yifei Li, Yijing Liu, Yiming Wang, Yingya Zhang, Yitong Huang, Yong Li, You Wu, Yu Liu, Yulin Pan, Yun Zheng, Yuntao Hong, Yupeng Shi, Yutong Feng, Zeyinzi Jiang, Zhen Han, Zhi-Fan Wu, and Ziyu Liu. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, 13(4): 600–612, 2004.

Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng-ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, et al. Qwen-image technical report. *arXiv preprint arXiv:2508.02324*, 2025.

Yifei Xia, Suhan Ling, Fangcheng Fu, Yujie Wang, Huixia Li, Xuefeng Xiao, and Bin Cui. Training-free and adaptive sparse attention for efficient long video generation. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15982–15993. IEEE, 2025.

Chenyang Xu, Dezhen Wang, Lin Chen, Kepeng Lin, and Hao Wang. Eigencache: Rethinking diffusion acceleration as covariance-optimal forecasting and submodular information allocation. In *Forty-third International Conference on Machine Learning*, 2026.

Jiazheng Xu, Xiao Liu, Yuchen Wu, Yuxuan Tong, Qinkai Li, Ming Ding, Jie Tang, and Yuxiao Dong. Imagereward: Learning and evaluating human preferences for text-to-image generation. *Advances in Neural Information Processing Systems*, 36:15903–15935, 2023.

Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. Cogvideox: Text-to-video diffusion models with an expert transformer. In *International Conference on Learning Representations*, volume 2025, pp. 83048–83077, 2025.

Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Fredo Durand, and William T Freeman. Improved distribution matching distillation for fast image synthesis. *Advances in neural information processing systems*, 37:47455–47487, 2024a.

Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Fredo Durand, William T Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6613–6623. IEEE, 2024b.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF conference on computer vision and pattern recognition*, pp. 586–595. IEEE, 2018.

Ruihang Zhang, Felix Taubner, Pooja Ravi, Kiriakos N Kutulakos, and David B Lindell. Proxypose: 6-dof pose tracking via video-to-video translation. *arXiv preprint arXiv:2607.06555*, 2026.

Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching. *arXiv preprint arXiv:2410.05317*, 2024.

A Appendix

A.1 Kalman Filter

A Kalman filter is the exact Bayesian estimator for a linear-Gaussian state-space model (Kalman, 1960). A hidden state ξ_j evolves linearly and is observed noisily,

$$\xi_{j+1} = A_j \xi_j + w_j, \quad w_j \sim \mathcal{N}(0, Q_j), \qquad y_j = \Pi\, \xi_j + e_j, \quad e_j \sim \mathcal{N}(0, R_j). \tag{13}$$

The filter maintains a Gaussian belief with mean μ and covariance P and alternates two operations. Between observations it *predicts*, $\mu \leftarrow A\mu$ and $P \leftarrow APA^\top + Q$, so the belief drifts forward and widens. When an observation arrives it *updates*, with gain $\mathcal{K} = P\Pi^\top (\Pi P \Pi^\top + R)^{-1}$,

$$\mu \leftarrow \mu + \mathcal{K}\, \underbrace{(y - \Pi\mu)}_{\text{innovation}}, \qquad P \leftarrow (I - \mathcal{K}\Pi)\, P. \tag{14}$$

The innovation is what the observation revealed that the model failed to predict. Comparing its realized magnitude against the predicted innovation variance $\Pi P \Pi^\top + R$ tests whether the assumed noise levels are right, which gives the filter a way to recalibrate itself online.

For a smooth trajectory with an unknown, slowly varying rate of change, the standard choice of state is a *local linear trend*: $\xi = (\text{level, rate})^\top$, where the level integrates the rate and the rate is itself driven by white noise. Both the transition and the process noise then depend only on the elapsed time Δ between successive observations,

$$A(\Delta) = \begin{pmatrix} 1 & \Delta \\ 0 & 1 \end{pmatrix}, \qquad Q(\Delta) = q \begin{pmatrix} \Delta^3/3 & \Delta^2/2 \\ \Delta^2/2 & \Delta \end{pmatrix}, \tag{15}$$

the latter being the integrated-Wiener covariance for a rate perturbed at intensity q.

Three properties make this a natural forecaster for equation 2. First, the predicted mean is the minimum-mean-squared-error estimate under the model and is informed by the entire observed history, and the gain shrinks the extrapolated rate whenever that rate is poorly determined; a finite difference instead commits fully to the last two observations and amplifies their noise. Second, irregular sampling is handled natively, since Δ enters A and Q directly, fixed-order difference formulas implicitly assume uniformly spaced observations, which a non-uniform compute schedule violates by construction. Third, and most useful here, the predictive covariance P is available *before* the next observation, so the filter states how wrong it expects to be. That turns the choice of $\mathcal{U}$ in from a heuristic into a thresholding decision.

It is worth distinguishing this from Wiener filtering, which also appears in the diffusion acceleration literature. A Wiener filter is the optimal *stationary* frequency response for a signal of fixed spectral power in fixed noise; the Kalman filter is its recursive, non-stationary counterpart, which tracks an evolving state and, crucially, its uncertainty. The filter itself is textbook material; what we contribute is the state-space model we place on the denoiser output and the way its covariance is calibrated and then spent (section 4).

B Details of the Band-specific Time Warp

This appendix expands on the time variable introduced in section 4.1.2. We derive the interpretation of $\lambda_j + \log p_\ell$ as the band's own signal-to-noise ratio, describe the mechanics of the warp, and collect the implementation details omitted from the main text.

B.1 The Warped Quantity is the Band's Own Log-SNR

Recall that a noisy latent is a linear interpolation between data and noise, $x_\sigma = (1 - \sigma)\, x_0 + \sigma\, \epsilon$ with $\epsilon \sim \mathcal{N}(0, I)$. Because the DCT is orthonormal, transforming white Gaussian noise yields white Gaussian noise, so every frequency coefficient of the noise term has unit variance and the noise contributes power σ^2 uniformly to every band. The data term is not uniform. Natural images concentrate energy at low frequencies, so the coefficients of band ℓ carry an average signal power p_ℓ

that decreases steeply with frequency. The signal-to-noise ratio of band ℓ at noise level σ is therefore approximately

$$\mathrm{SNR}_\ell(\sigma) \;=\; \frac{(1-\sigma)^2\, p_\ell}{\sigma^2} \qquad \Longrightarrow \qquad \log \mathrm{SNR}_\ell(\sigma) \;=\; \underbrace{2\log\!\left(\tfrac{1-\sigma}{\sigma}\right)}_{\lambda_j} \;+\; \log p_\ell. \tag{16}$$

The argument of the sigmoid in equation 8 is thus exactly the band's own log-SNR. It crosses zero when $\mathrm{SNR}_\ell = 1$, the point at which the band's signal emerges from the noise. Since p_ℓ is large for low bands and small for high bands, low bands cross this threshold early in denoising (at large σ) and high bands cross it late, which formalizes the observation that different bands do most of their changing in different phases.

B.2 Mechanics of the Warp

The sigmoid maps the band's log-SNR to $(0, 1)$ and its derivative is largest at the zero crossing and decays symmetrically away from it. Consequently, the warped time $t_\ell(j) = \mathrm{sigmoid}\left((\lambda_j + \log p_\ell)/T_w\right)$ advances fastest over the steps surrounding the band's emergence and barely moves while the band is still buried in noise or already resolved. The temperature T_w sets the width of this active window in log-SNR units; a small T_w concentrates the time variable tightly around the crossing, while a large T_w approaches a plain rescaling of λ_j. We use $T_w = 2$ for all bands and both models.

Since σ_j decreases monotonically over sampling, λ_j increases monotonically, and both the sigmoid and the final rescaling are monotone maps. Each s_ℓ is therefore a strictly increasing function of the step index, a requirement for a valid time variable. The operator $\mathrm{rescale}_{[-1,1]}$ is the affine map taking the smallest and largest values of t_ℓ over the N steps of the run to -1 and 1 respectively, so every band's time variable spans the same interval regardless of where its active window lies. This normalization has a second use. The interval $[-1, 1]$ is the natural domain of Chebyshev polynomials, so the polynomial-fit ablations of section 5.3 can operate directly in the same coordinate without further transformation.

The warp interacts with the forecaster through the increments $\Delta_\ell = s_\ell(j{+}1) - s_\ell(j)$, which serve as the effective step sizes of the per-band Kalman filters. During a band's active window the increments are large, so the filter expects more change per step, injects more process noise, and damps its trend estimate faster; outside the window the increments are small and the filter treats the band as nearly static. The warp thus lets a single, simple dynamics model adapt its behavior to each band and phase without any per-step logic.

B.3 Implementation Details

Two practical points matter for reproducing the coordinate exactly. First, modern rectified-flow pipelines apply a resolution-dependent shift to the sigma schedule, so the realized σ_j differ from the nominal linspace values. SpecKCast rebuilds the exact shifted schedule the pipeline will use before sampling begins and computes all $s_\ell(j)$ from it, so the controller's time variables line up with the transformer calls at any resolution and step count. Second, the endpoints of the schedule are clamped away from $\sigma = 0$ and $\sigma = 1$ before evaluating λ_j to avoid infinities; the clamping affects only the first and last values of t_ℓ and is absorbed by the rescaling. The band powers p_ℓ are measured once during calibration (section B.4) as the average per-coefficient power of each band of the implied clean image $\hat{x}_0 = x_j - \sigma_j Y_j$ over the trailing portion of the harvested trajectories, where $\hat{x}_0$ has converged. Since the warp is a fixed elementwise function of the schedule, the entire coordinate system costs a few scalar operations per run and is computed once before sampling.

B.4 Band-Power Calibration

SpecKCast uses p_ℓ to account for differences in signal magnitude across frequency bands. We estimate these values once from a small calibration set $\mathcal{C}$ by running the unaccelerated model and recording the latent x_j, model output Y_j, and noise level σ_j at every denoising step.

Under the rectified-flow interpolation,

$$x_j = (1-\sigma_j)x_0 + \sigma_j \epsilon = x_0 + \sigma_j(\epsilon - x_0). \tag{17}$$

Because the model predicts the velocity $Y_j \approx \epsilon - x_0$, it provides the clean-sample estimate

$$\widehat{x}_{0,j} = x_j - \sigma_j Y_j. \tag{18}$$

We estimate power using the final denoising steps, where σ_j is small. In particular, if

$$Y_j = (\epsilon - x_0) + \delta_j, \tag{19}$$

then

$$\widehat{x}_{0,j} = x_0 - \sigma_j \delta_j, \tag{20}$$

so velocity-prediction errors have less influence on $\widehat{x}_{0,j}$ near the end of denoising.

We apply the same channel-wise orthonormal DCT used by SpecKCast,

$$X_{0,j} = \mathrm{DCT}_{2\mathrm{D}}(\widehat{x}_{0,j}). \tag{21}$$

Let Ω_ℓ contain the spatial-frequency indices belonging to band ℓ. We estimate its signal power by averaging the squared DCT coefficients over calibration samples, selected denoising steps, channels, and frequencies:

$$p_\ell = \frac{1}{|\mathcal{C}||\mathcal{J}|C|\Omega_\ell|} \sum_{n\in\mathcal{C}} \sum_{j\in\mathcal{J}} \sum_{c=1}^{C} \sum_{(u,v)\in\Omega_\ell} \left| X^{(n)}_{0,j,c,u,v} \right|^2, \tag{22}$$

where $\mathcal{J}$ denotes the selected final denoising steps.

The connection between p_ℓ and the time warp follows from the band-level signal-to-noise ratio. Because an orthonormal DCT preserves isotropic Gaussian noise, every transformed noise coefficient has unit variance. The approximate SNR of band ℓ at noise level σ_j is therefore

$$\mathrm{SNR}_{\ell,j} \approx p_\ell \frac{(1-\sigma_j)^2}{\sigma_j^2}. \tag{23}$$

Taking its logarithm gives

$$\log \mathrm{SNR}_{\ell,j} = \lambda_j + \log p_\ell, \tag{24}$$

which is precisely the quantity used to construct the band-specific time coordinate. Thus, p_ℓ shifts the global log-SNR according to the signal strength of each band, allowing the resulting coordinate to reflect when that band transitions from noise-dominated to signal-dominated.

In our implementation, the calibration uses 12 prompts, 50 denoising steps, and the final 25 steps of each trajectory. These values are fixed across prompts during inference. Calibration modifies no model parameters and introduces no additional model evaluations at inference time.

Calibration-free alternatives. Offline calibration is not required to apply SpecKCast. The simplest alternative is to disable the band-specific warp and use rescaled log-SNR for every band,

$$s_\ell(j) = \mathrm{rescale}_{[-1,1]}(\lambda_j). \tag{25}$$

Forecasting remains valid, but all bands then share the same temporal coordinate. Consequently, elapsed time no longer reflects their different signal-to-noise transitions, removing the intended band-specific control of extrapolation, damping, and uncertainty. This does not necessarily imply lower generation quality and is evaluated through the no-warp ablation.

Alternatively, p_ℓ can be estimated online from the first K computed outputs,

$$p_\ell = \frac{1}{K d_\ell} \sum_{j=1}^{K} \sum_{k\in\mathcal{B}_\ell} Z^2_{j,k}, \tag{26}$$

where $d_\ell = |\mathcal{B}_\ell|$. When these observations are collected during the mandatory warm-up, this approach requires no additional transformer evaluations. However, the estimate is based on fewer observations, can vary across prompts, and measures model-output power rather than clean-signal power. It may therefore yield less stable time coordinates than offline calibration.

B.5 Rationale for Band-Specific Time Reparameterization

The denoising process follows one global sequence of steps j and noise levels σ_j. SpecKCast does not assign different physical denoising times to different frequency bands. Instead, it represents the same global step using a different monotonic coordinate for each band:

$$s_\ell(j) = g_\ell(\lambda_j), \tag{27}$$

where g_ℓ is the warp associated with band ℓ. Because each g_ℓ is monotonic, the ordering of the denoising steps is preserved:

$$j_1 < j_2 \quad \Longrightarrow \quad s_\ell(j_1) < s_\ell(j_2). \tag{28}$$

Thus, every band follows the same denoising direction and reaches the same target step, although the effective distance between two steps can differ across bands. For a skip from step j to step k, band ℓ experiences the effective time interval

$$\Delta_\ell(j, k) = s_\ell(k) - s_\ell(j). \tag{29}$$

This interval can be large for a band expected to change substantially during the skip and small for a band expected to remain stable. Consequently, the forecaster can make a larger update for an active band and a more conservative prediction for a stable one. This principle is not specific to the Kalman filter. Any forecasting model that uses time as an input, including linear extrapolation and polynomial fitting, can operate in a band-specific coordinate. The warp used by SpecKCast is

$$g_\ell(\lambda) = \mathrm{rescale}_{[-1,1]}\left[\mathrm{sigmoid}\left(\frac{\lambda + \log p_\ell}{T_w}\right)\right]. \tag{30}$$

For $p_\ell > 0$ and $T_w > 0$, this mapping is monotonic in λ. The subsequent affine rescaling also preserves monotonicity. All bands therefore share the endpoints -1 and 1, while their progression between these endpoints can differ. At a target step k, each band is forecast at its corresponding coordinate $s_\ell(k)$. The resulting coefficients $\widehat{Z}_k^{(\ell)}$ are combined into the complete frequency representation $\widehat{Z}_k$ and transformed back to the spatial domain:

$$\widehat{Y}_k = \mathrm{IDCT}_{2\mathrm{D}}(\widehat{Z}_k). \tag{31}$$

All reconstructed coefficients therefore represent the same physical denoising step k. The band-specific coordinates only determine how their temporal evolution is modeled; they do not modify the noise schedule, sampler, or denoising trajectory. This reparameterization provides an inductive bias rather than a guarantee of improved forecasting. It assumes that matching each band's effective clock to its expected evolution makes its trajectory easier to predict. Its contribution is therefore evaluated against a shared-clock or unwarped log-SNR baseline.

B.6 Rationale for the Kalman Dynamics and Process Noise

The transition matrix $A_{\ell,j}$ and process-noise covariance $Q_{\ell,j}$ serve different purposes. The former describes the expected evolution of a coefficient, while the latter represents deviations from that evolution.

For coefficient i in band ℓ, the state is

$$h_{i,j} = (m_{i,j}, \nu_{i,j}), \tag{32}$$

where $m_{i,j}$ is the coefficient value and $\nu_{i,j}$ is its temporal derivative. Over the interval $\Delta_{\ell,j}$, we advance the value using its derivative and attenuate the derivative by

$$\rho_{\ell,j} = \exp(-\Delta_{\ell,j}/\tau_\ell). \tag{33}$$

These operations give

$$A_{\ell,j} = \begin{bmatrix} 1 & \Delta_{\ell,j} \\ 0 & \rho_{\ell,j} \end{bmatrix}. \tag{34}$$

The upper-right entry integrates the derivative over the interval, while the lower-right entry prevents old changes from being extrapolated indefinitely. As the interval grows, $\rho_{\ell,j}$ decreases and the prediction relies less on the previous derivative.

The deterministic transition cannot capture all changes in the coefficient trajectory. We model these deviations as continuous random perturbations to the temporal derivative. If their intensity is $q_{\ell,j}$, the accumulated state perturbation over an interval $\Delta_{\ell,j}$ has covariance

$$Q_{\ell,j} = q_{\ell,j} \int_0^{\Delta_{\ell,j}} \begin{bmatrix} (\Delta_{\ell,j} - u)^2 & \Delta_{\ell,j} - u \\ \Delta_{\ell,j} - u & 1 \end{bmatrix} du, \tag{35}$$

which evaluates to

$$Q_{\ell,j} = q_{\ell,j} \begin{bmatrix} \Delta_{\ell,j}^3/3 & \Delta_{\ell,j}^2/2 \\ \Delta_{\ell,j}^2/2 & \Delta_{\ell,j} \end{bmatrix}. \tag{36}$$

The lower-right term describes uncertainty accumulated in the temporal derivative. Integrating that uncertainty produces the cross-covariance terms and the cubic growth of uncertainty in the coefficient value. Consequently, longer forecast intervals result in larger covariance.

The scalar $q_{\ell,j}$ controls the magnitude of this uncertainty. Bands with greater temporal activity receive larger process intensity, allowing their observations to produce stronger corrections. Since coefficients in the same band share $\Delta_{\ell,j}$, $\rho_{\ell,j}$, and $q_{\ell,j}$, they also share $A_{\ell,j}$, $Q_{\ell,j}$, and the resulting Kalman covariance, while retaining separate state estimates.

This construction has intuitive limiting behavior. As $\Delta_{\ell,j} \to 0$, $A_{\ell,j} \to I$ and $Q_{\ell,j} \to 0$, since little evolution or uncertainty is expected over a negligible interval. For longer intervals, the derivative is increasingly attenuated while process uncertainty grows.

This transition is a stepwise discrete approximation rather than the exact discretization of a continuous damped system. It first advances the coefficient using the derivative at the beginning of each interval and then attenuates that derivative. Because the update is applied successively over all intervening denoising steps, the damping compounds across a multi-step skip.

Bounded accumulation across skipped steps. The damping acts through repeated propagation over the intervening denoising steps, rather than within a single large update. To illustrate this, consider K skipped steps with a constant increment Δ_ℓ and $\rho_\ell = \exp(-\Delta_\ell/\tau_\ell)$. Starting from $\nu_{i,0}$, the temporal derivative after k steps is

$$\nu_{i,k} = \rho_\ell^k \nu_{i,0}. \tag{37}$$

The total change in the predicted coefficient value is therefore

$$m_{i,K} - m_{i,0} = \sum_{k=0}^{K-1} \Delta_\ell \rho_\ell^k \nu_{i,0} = \Delta_\ell \frac{1 - \rho_\ell^K}{1 - \rho_\ell} \nu_{i,0}. \tag{38}$$

Consequently,

$$|m_{i,K} - m_{i,0}| \le \frac{\Delta_\ell}{1 - \rho_\ell} |\nu_{i,0}|. \tag{39}$$

Thus, the contribution of the initial temporal derivative remains bounded as the skip length increases. In contrast, without damping, $\rho_\ell = 1$ and the displacement grows linearly as $K\Delta_\ell \nu_{i,0}$. Moreover,

$$\lim_{\Delta_\ell \to 0} \frac{\Delta_\ell}{1 - \exp(-\Delta_\ell/\tau_\ell)} = \tau_\ell, \tag{40}$$

so the bound approaches $\tau_\ell |\nu_{i,0}|$ as the discretization is refined.

For nonuniform increments, the derivative attenuation still composes exactly:

$$\prod_{r=1}^{K} \rho_{\ell,r} = \exp\left(-\frac{\sum_{r=1}^{K} \Delta_{\ell,r}}{\tau_\ell} \right). \tag{41}$$

The accumulated value update depends on the underlying denoising grid, but the derivative continues to decay with the total elapsed band time. This bound concerns the deterministic extrapolation induced by the current derivative; it does not by itself bound the total forecasting error.

Process-noise covariance. The deterministic transition captures the expected evolution of each coefficient, but the actual trajectory may deviate from this model. We represent these deviations by adding white noise to the temporal derivative. Ignoring damping momentarily, this corresponds to

$$dm = \nu\, ds, \qquad d\nu = \sqrt{q_{\ell,j}}\, dW_s, \tag{42}$$

where W_s is a Wiener process and $q_{\ell,j}$ controls the intensity of the perturbation.

A perturbation occurring at time u affects the temporal derivative directly and the coefficient value through integration over the remaining interval. Its accumulated covariance over an interval $\Delta_{\ell,j}$ is therefore

$$Q_{\ell,j} = q_{\ell,j} \int_0^{\Delta_{\ell,j}} \begin{bmatrix} (\Delta_{\ell,j} - u)^2 & \Delta_{\ell,j} - u \\ \Delta_{\ell,j} - u & 1 \end{bmatrix} du, \tag{43}$$

which gives

$$Q_{\ell,j} = q_{\ell,j} \begin{bmatrix} \Delta_{\ell,j}^3/3 & \Delta_{\ell,j}^2/2 \\ \Delta_{\ell,j}^2/2 & \Delta_{\ell,j} \end{bmatrix}. \tag{44}$$

The lower-right entry describes uncertainty accumulated in the temporal derivative. Integrating this uncertainty produces the quadratic cross-covariance and cubic value-variance terms. Thus, longer forecast intervals introduce greater uncertainty into both state components. The resulting matrix is positive definite for $q_{\ell,j} > 0$ and $\Delta_{\ell,j} > 0$, since

$$\det(Q_{\ell,j}) = \frac{q_{\ell,j}^2 \Delta_{\ell,j}^4}{12} > 0. \tag{45}$$

eq. (44) is the standard integrated-white-acceleration covariance for an undamped system. We deliberately combine it with the stepwise damped transition as a simple uncertainty approximation. To understand this choice, consider the exact continuous damped model with $\rho_{\ell,j} = \exp(-\Delta_{\ell,j}/\tau_\ell)$. Its diagonal process variances are

$$Q_{11}^{\text{damp}} = q_{\ell,j}\tau_\ell^2 \left[\Delta_{\ell,j} - 2\tau_\ell(1-\rho_{\ell,j}) + \frac{\tau_\ell}{2}(1-\rho_{\ell,j}^2)\right], \tag{46}$$

$$Q_{22}^{\text{damp}} = \frac{q_{\ell,j}\tau_\ell}{2}(1-\rho_{\ell,j}^2). \tag{47}$$

Because

$$\tau_\ell(1 - e^{-x/\tau_\ell}) \le x, \qquad e^{-x/\tau_\ell} \le 1, \tag{48}$$

the undamped form satisfies

$$Q_{11}^{\text{damp}} \le q_{\ell,j}\frac{\Delta_{\ell,j}^3}{3}, \qquad Q_{22}^{\text{damp}} \le q_{\ell,j}\Delta_{\ell,j}. \tag{49}$$

Our choice therefore overestimates the marginal variance growth relative to the exact damped model. This is the conservative direction for uncertainty-based scheduling, since it favors earlier model evaluation rather than overconfident forecasting. For small adjacent intervals, the damped and undamped forms agree to leading order. Any remaining mismatch is absorbed by the calibrated and online-adapted process intensity $q_{\ell,j}$.

C Details of the Uncertainty-Aware Adaptive Scheduler

This appendix specifies the machinery behind Section 4.3: the covariance recursion used to evaluate the risk, its calibrated noise and online correction, the exact decision rule with all safeguards, and the fixed-schedule variant used in our ablations. We also illustrate how evaluation schedules vary with the threshold and prompt.

C.1 The Risk Sidecar

The risk of Equation (12) is evaluated by a dedicated covariance recursion, which we call the *sidecar*, that runs alongside the mean filter of section 4.2 but is decoupled from it in two ways. First, it operates in *relative* units, tracking forecast error as a fraction of each band's power, which makes

its outputs comparable across bands and generations. Second, its process noise is not the moment-matched dynamics noise of the mean filter but a table $Q^{\mathrm{sc}}_{\ell,w}$ calibrated directly against the forecaster's realized errors (Section B.4), so the sidecar answers the question the scheduler actually asks, namely how wrong the forecast tends to be after a given stretch of skipped steps. The index w ranges over four step windows partitioning the schedule, allowing the noise to differ between early, middle, and late denoising.

The sidecar maintains one 2×2 covariance per band with entries $(\tilde{p}_{11}, \tilde{p}_{12}, \tilde{p}_{22})$ and propagates them with the same integrated-noise structure as

$$\begin{aligned} \tilde{p}_{11} &\leftarrow \tilde{p}_{11} + 2\Delta_\ell\, \tilde{p}_{12} + \Delta_\ell^2\, \tilde{p}_{22} + Q^{\mathrm{sc}}_{\ell,w}\, \Delta_\ell^3/3, \\ \tilde{p}_{12} &\leftarrow \rho_{\mathrm{sc}}\big(\tilde{p}_{12} + \Delta_\ell\, \tilde{p}_{22}\big) + Q^{\mathrm{sc}}_{\ell,w}\, \Delta_\ell^2/2, \\ \tilde{p}_{22} &\leftarrow \rho_{\mathrm{sc}}^2\, \tilde{p}_{22} + Q^{\mathrm{sc}}_{\ell,w}\, \Delta_\ell, \end{aligned} \tag{50}$$

with a fixed memory factor $\rho_{\mathrm{sc}} = 0.9$ per step. At every computed step the sidecar conditions on the fact that the value has just been observed exactly. The value uncertainty and its cross term collapse to zero and the trend uncertainty is reduced to its posterior,

$$\tilde{p}_{11} \leftarrow 0, \qquad \tilde{p}_{12} \leftarrow 0, \qquad \tilde{p}_{22} \leftarrow \tilde{p}_{22} - \tilde{p}_{12}^2/\tilde{p}_{11}, \tag{51}$$

so risk restarts from zero after every transformer call and re-accumulates over the following skips. The quantity $p_{11}^{(\ell)}$ in Equation (12) refers to $\tilde{p}_{11}$ propagated by Equation (50).

Like the mean filter, the sidecar corrects itself online. At each computed step the realized squared innovation of the mean forecaster, averaged per band and normalized by band power, is compared with the sidecar's predicted variance $\tilde{p}_{11}$; their ratio, smoothed by an exponential moving average with rate 0.2 and clamped to $[0.1, 10]$, forms a per-band scale $\tilde{\kappa}_\ell$ that multiplies the band's contribution to the risk. Generations that turn out harder to forecast than the calibration data therefore raise their own risk and are scheduled more conservatively, and vice versa. The full risk is

$$R(j) \;=\; \sum_{\ell} \tilde{\kappa}_\ell\; \gamma_{\ell,w(j)}\; \tilde{p}_{11}^{(\ell)}(j), \tag{52}$$

which reduces to Equation (12) with the online scale absorbed into the weight. The perceptual weights $\gamma_{\ell,w}$ are calibrated by perturbation, injecting band-limited errors into the output at different phases of sampling and measuring the LPIPS change of the final image (Section B.4); when such weights are unavailable, the band dimensionalities d_ℓ are used as a neutral fallback.

C.2 Decision Rule

After each computed step j, the sidecar is rolled forward *predictively*, applying Equation (50) over a horizon of up to 10 future steps without changing its state, and the resulting risks $R(j{+}1), R(j{+}2), \ldots$ are compared against the threshold. The next computed step is placed at the first predicted crossing $R > \varepsilon^2$; all steps before it are forecast. Three safeguards apply. The first 3 steps of sampling are always computed, giving the filters real observations to establish values and trends; the final step is always computed, so the image is finished by a true model evaluation; and the number of consecutive skips is capped at 10 regardless of the predicted risk. The threshold is set once per compute budget by a closed-loop calibration that replays the scheduler on held-out trajectories and bisects ε until the average number of transformer calls meets the target (Section B.4); at inference, the realized count then varies around this target from generation to generation.

D Extended Qualitative Comparisons

We extend the comparisons in Sections 5.1 and 5.2 with additional Qwen-Image, HunyuanVideo, and Wan2.1-14B examples. Each panel reports its evaluation count and measured speedup relative to the 50-step reference. These examples illustrate the visual effects of acceleration; aggregate results are reported in Tables 1 and 2.

Why SpecKCast preserves the reference more closely. As discussed in Section 2, cached features become outdated over long skips, while feature-space extrapolation does not distinguish the temporal dynamics of different spatial frequencies. SpecKCast forecasts the sampler's final velocity coefficient by coefficient, using band-specific time coordinates to account for these differences. Damped trends limit overshooting, fresh evaluations correct the forecast, and uncertainty-aware scheduling allocates computation when the predicted error grows. The ablations in Table 3 support the contributions of frequency decomposition, time warping, and adaptive scheduling; the examples below illustrate the combined method.

Qwen-Image: color, texture, and object structure. In Figure 7, TaylorSeer and HiCache give the strawberry scene a strong blue cast and darken the fox scene. TeaCache changes the basket's contents and introduces a second astronaut, while SeaCache adds an extra horse leg and smooths the fox's fur. Spectrum introduces grain and distorts the rider. SpecKCast more closely retains the reference lighting, object arrangement, and recognizable details, with 6.19–$6.21\times$ speedup at 8 NFE and $4.97\times$ at 10 NFE. The gain is therefore visible in both overall appearance and local structure, at lower latency than the displayed baselines.

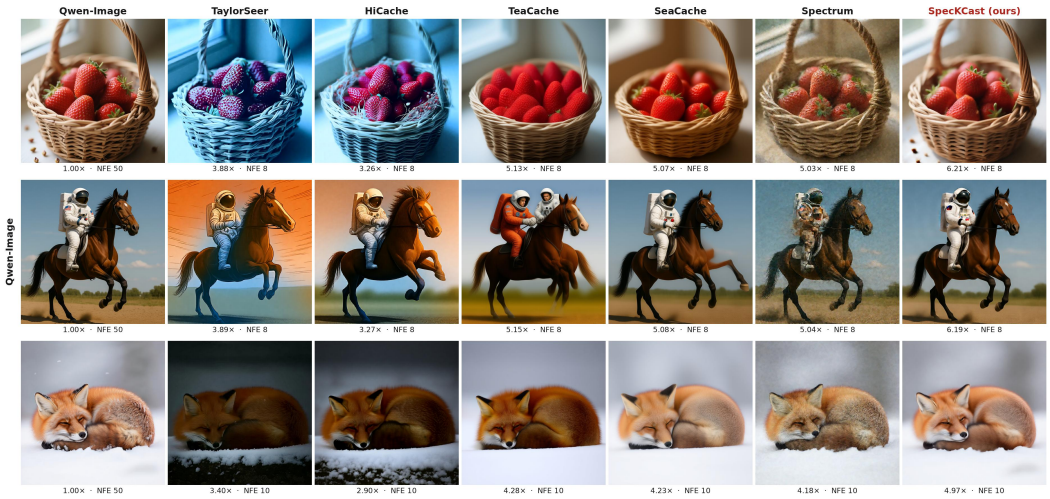

Figure 7: **Additional Qwen-Image comparisons.** Rows show strawberries in a basket, an astronaut riding a horse, and a fox in snow. The reference uses 50 evaluations; all accelerated methods use 8 NFE in the first two rows and 10 NFE in the last. SpecKCast more closely preserves the reference colors and object structure, while the baselines exhibit color shifts, altered content, or degraded texture.

HunyuanVideo: subject detail and geometry. In Figure 8, FORA and TaylorSeer blur the face, TeaCache changes the action and produces overlapping hand and face structure, and HiCache changes the subject's appearance. SpecKCast produces a clearer face and retains the bottle in hand across the displayed frames at 10 NFE and $4.27\times$ speedup, although the subject's appearance still differs from the reference. In Figure 9, TaylorSeer and Spectrum introduce an extra wheel-like structure between the bicycle's two wheels. SpecKCast preserves the two-wheel geometry with fewer evaluations, though it omits the reference's front basket.



Figure 8: **HunyuanVideo: "a bottle and a chair."** Three displayed frames per method reveal differences in facial detail, pose, and bottle placement. SpecKCast uses 10 NFE ($4.27\times$), compared with 10–11 NFE (3.85–$4.24\times$) for the baselines, and avoids the severe facial blur visible in FORA and TaylorSeer.

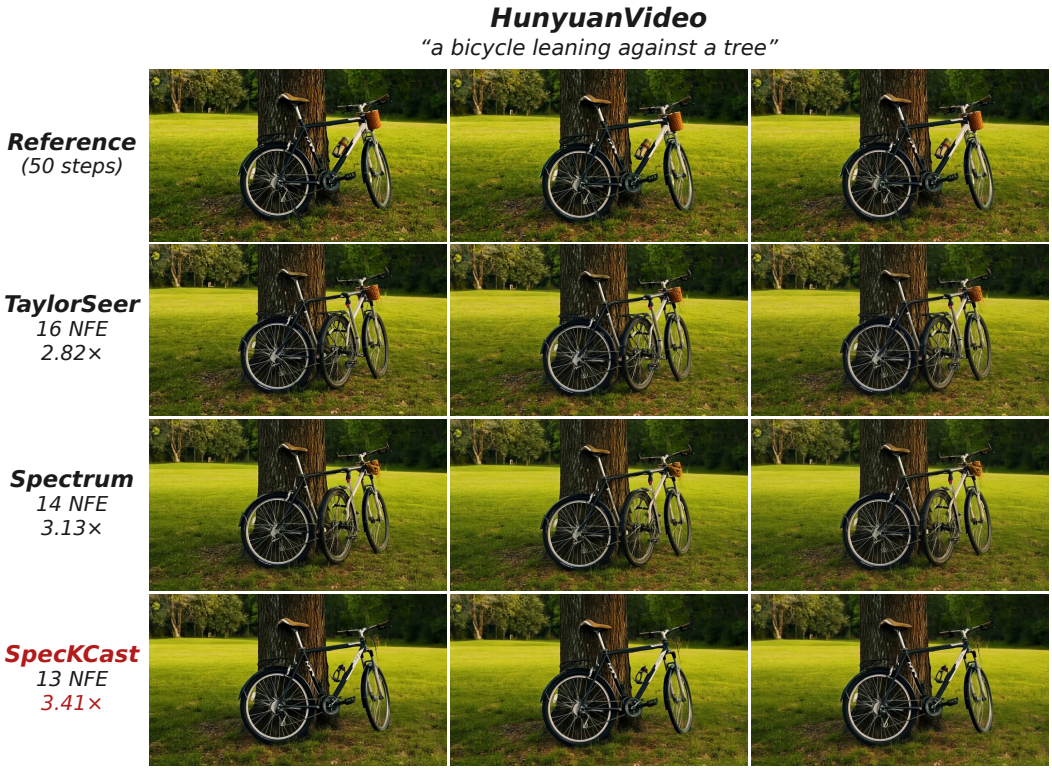

Figure 9: **HunyuanVideo: "a bicycle leaning against a tree."** TaylorSeer (16 NFE, 2.82×) and Spectrum (14 NFE, 3.13×) introduce an additional wheel-like structure. SpecKCast (13 NFE, 3.41×) better preserves the main bicycle geometry across the displayed frames, although the front basket is missing.

Wan2.1-14B: appearance and scene structure at 8 NFE. In Figure 10, HiCache and TaylorSeer produce strong saturation and blurred boundaries. FORA and TeaCache retain the layout but soften the panda's detail, while SeaCache changes its appearance and the tableware at 7 NFE. SpecKCast retains the reference composition and drinking poses at $5.61\times$ speedup. In Figure 11, TaylorSeer severely distorts the boat and skyline, and Spectrum alters the Eiffel Tower's shape. SpecKCast retains a more faithful landmark silhouette and river layout at the same 8 NFE and $5.65\times$ speedup as Spectrum.

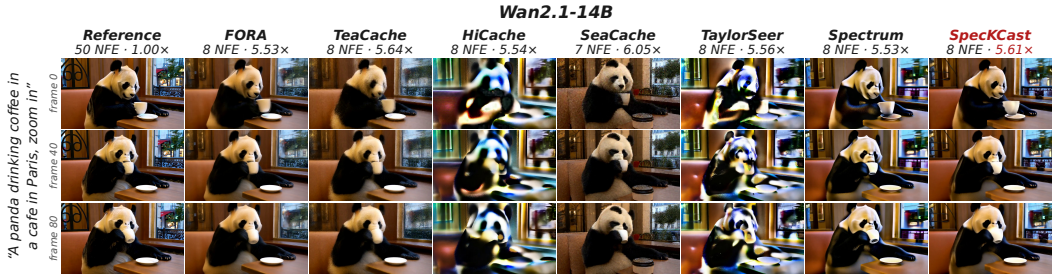

Figure 10: **Wan2.1-14B: a panda drinking coffee with a zoom-in.** Frames 0, 40, and 80 show subject appearance and scene layout. All accelerated methods use 8 NFE except SeaCache (7 NFE). SpecKCast achieves $5.61\times$ speedup while avoiding the strong color and boundary artifacts of HiCache and TaylorSeer; SeaCache is faster at $6.05\times$ but changes the scene more substantially.

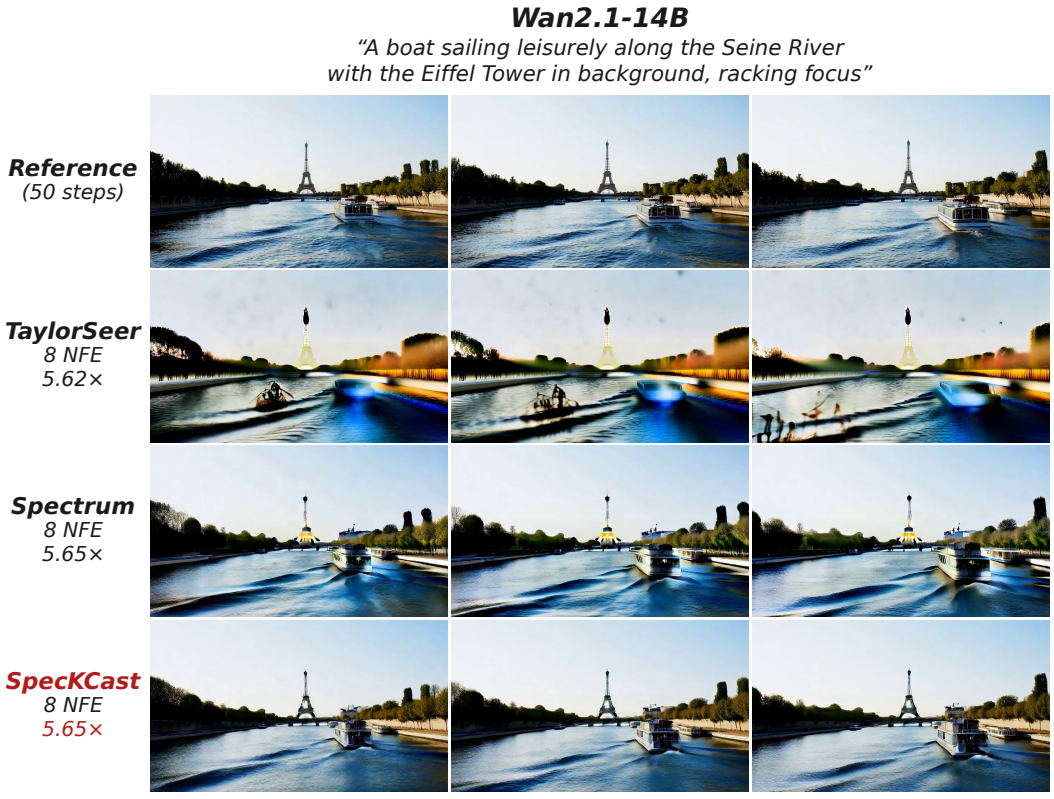

Figure 11: **Wan2.1-14B: a boat on the Seine with the Eiffel Tower in the background.** All three accelerated methods use 8 NFE, with speedups of $5.62\times$ (TaylorSeer) and $5.65\times$ (Spectrum and SpecKCast). SpecKCast more closely preserves the tower's silhouette and scene layout across the displayed frames.

D.1 Threshold-Controlled Evaluation Budgets

Figure 12 illustrates how the threshold controls the evaluation budget for a fixed prompt. Raising the threshold permits longer skips and fewer evaluations, while lowering it leads to more frequent evaluations. The schematics here and in Section D.2 isolate the threshold mechanism; the complete scheduler also applies the safeguards in Section C.2.

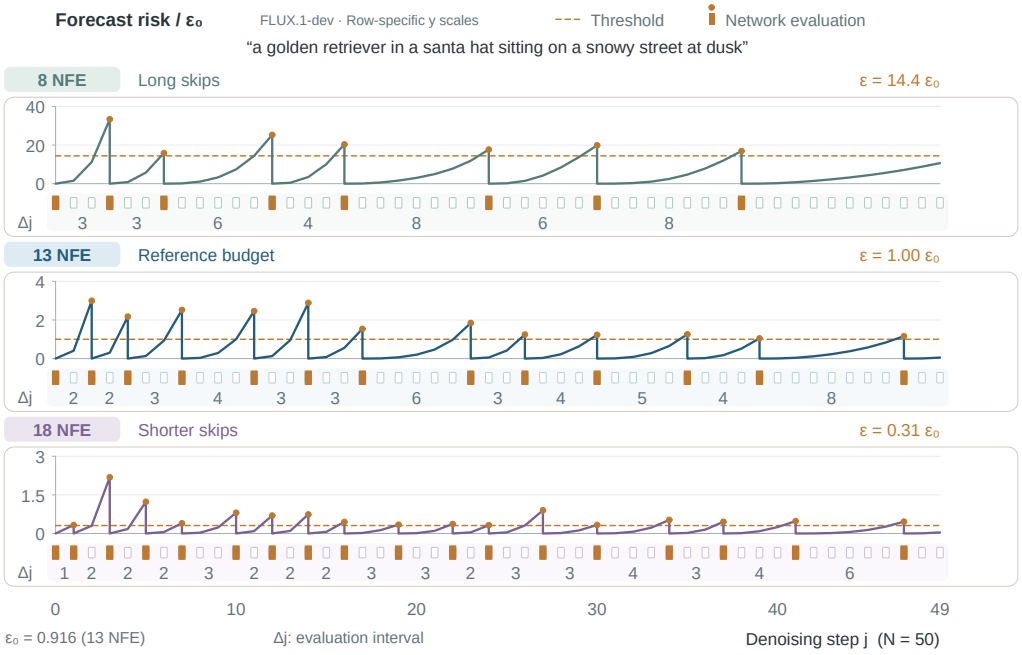

Figure 12: **Threshold-controlled scheduling for a fixed prompt.** Schematic risk traces over $N = 50$ denoising steps for the prompt shown, using the FLUX.1-dev threshold calibration. From top to bottom, thresholds are approximately $14.4\varepsilon_0$, ε_0, and $0.31\varepsilon_0$ for illustrative schedules with 8, 13, and 18 network function evaluations (NFE), where $\varepsilon_0 = 0.916$. The corresponding absolute thresholds are 13.20, 0.916, and 0.283. All curves show schematic $R(j)/\varepsilon_0$ on the individually labeled linear vertical scales; each source trace is rescaled with its threshold while retaining its evaluation schedule. Dashed orange lines indicate thresholds; orange dots and filled boxes mark evaluations, while open boxes mark forecasted steps. The labels Δj give the intervals between consecutive evaluations. Lower thresholds produce more frequent evaluations and shorter skips.

D.2 Prompt-Dependent Evaluation Schedules

Figure 13 illustrates how the placement of evaluations can change across prompts even when the total evaluation count is held fixed. The vase example concentrates evaluations early, while the night-market example allocates more evaluations toward the end of denoising. Together with the threshold comparison for a fixed prompt in Figure 12, these schematics illustrate the scheduler's control over both the number and timing of evaluations.

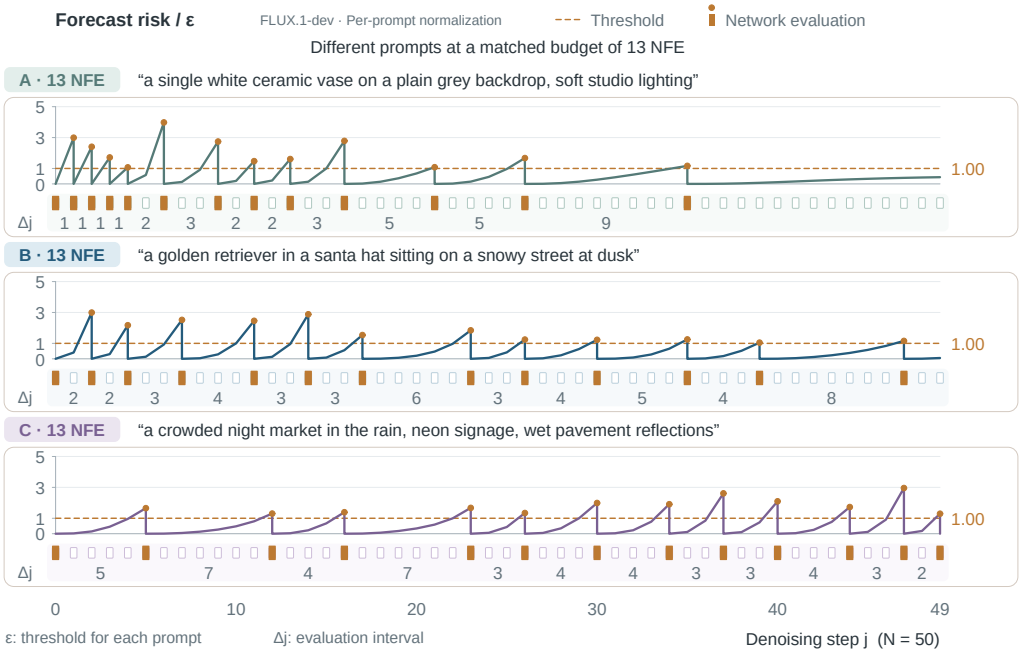

Figure 13: **Prompt-dependent scheduling at a fixed evaluation count.** Three illustrative schedules each use 13 network function evaluations (NFE) over $N = 50$ denoising steps. Rows correspond to the vase, golden-retriever, and night-market prompts shown. Each risk curve is normalized by its own prompt's threshold, $R(j)/\varepsilon$, so all dashed orange threshold lines lie at 1; absolute risk magnitudes are not directly comparable across rows. Orange dots and filled boxes mark evaluations, open boxes mark forecasted steps, and Δj denotes the interval between consecutive evaluations. The same evaluation count permits different allocations of computation along the denoising trajectory.

D.3 Fixed-Schedule Variant

As a deterministic alternative, we support a fixed schedule with quadratically growing gaps, computing steps

$$j_r = \lfloor \alpha\, r(r+1)/2 \rfloor, \qquad r = 1, 2, \ldots, \tag{53}$$

together with the warmup steps $\{0, 1, 2\}$ and the final step, where α is chosen by a simple search so that the total number of computed steps matches the target budget. The spacing is dense early in sampling, where the output changes fastest, and sparse late, mirroring the qualitative shape that the adaptive scheduler discovers on typical generations. The two schedulers offer a practical trade-off explored in our experiments; the fixed variant is preferable when a deterministic compute cost is required, while the adaptive variant reallocates calls between easy and hard generations at matched average cost.

E Additional Spatial-frequency Trajectory Analysis

We supplement the pooled FLUX.1-dev trajectories in Figure 2 with complementary views and individual generations, and extend the analysis to Qwen-Image. Both collections use the same 12 prompts, with one generation per prompt and model. Figure 14 shows the planar projections and scores against denoising step corresponding to Figure 2; Figure 15 provides all three pooled views for Qwen-Image. The remaining figures report every generation separately. Together, these visualizations examine how temporal evolution differs across spatial-frequency bands and how the patterns vary across prompts and models.

E.1 Measurement Setup

For both models, each generation uses a resolution of 1024×1024, 50 denoising steps indexed from 0 to 49, and full transformer computation at every step. We use zero-based prompt IDs: prompt $p \in \{0, \dots, 11\}$ uses generation seed p. The prompt texts are reproduced in the individual figure captions.

FLUX.1-dev. The FLUX collection uses guidance scale 3.5. We retain 512 fixed sampled entries of the full velocity and 128, 512, and 512 DCT coefficients from radial bands B_0, B_3, and B_6, respectively. Coefficient sampling uses seed 1234, with at most 512 entries per band. The pooled analysis combines the measurements from all 12 generations, while each individual analysis uses only its own generation.

Qwen-Image. We record the velocity passed to the scheduler after classifier-free guidance at scale 4 and the pipeline's norm rescaling. For conditional and unconditional predictions $c_{j,t}$ and $u_{j,t}$ at packed token t, the recorded output is

$$\widetilde{v}_{j,t} = u_{j,t} + 4(c_{j,t} - u_{j,t}), \qquad v_{j,t} = \widetilde{v}_{j,t} \, \frac{\|c_{j,t}\|_2}{\|\widetilde{v}_{j,t}\|_2}, \tag{54}$$

where the negative prompt is empty and each norm is taken over the packed token's 64 channels. Unpacking this velocity gives $Y_j \in \mathbb{R}^{16 \times 128 \times 128}$.

For Qwen-Image, we apply an orthonormal two-dimensional DCT-II independently to each channel. For spatial-frequency indices (a, b), the normalized radius is

$$r(a, b) = \frac{\sqrt{(a/128)^2 + (b/128)^2}}{\sqrt{2}}, \qquad a, b \in \{0, \dots, 127\}. \tag{55}$$

The seven radial bands have edges $[0, 1/64, 1/32, 1/16, 1/8, 1/4, 1/2, 1]$. The four Qwen panels visualize 512 fixed sampled entries of the full unpacked velocity, all 128 coefficients of B_0, and 512 fixed sampled coefficients each from B_3 and B_6. The sampled indices are shared across steps and prompts. In both models, the full-velocity panel summarizes sampled entries of the spatial-domain output, while the band panels summarize DCT coefficients.

E.2 PCA Construction and Display

Both models use the same pooling and normalization procedure, applied independently to each model. For each prompt and panel, we arrange the recorded values into a coefficient-by-step matrix and divide it by its Frobenius norm. For a pooled view, the 12 normalized matrices are concatenated along the coefficient axis and transposed, giving a matrix with 50 observations and $12d$ features, where d is the number of retained entries in that panel. We center each feature across steps and project onto its first two principal components. Thus, each pooled curve summarizes joint temporal variation across 12 generations. For the individual views, the same procedure is applied to each prompt separately, giving a distinct PCA basis for each prompt and panel.

Within each panel, both score sequences are divided by their joint maximum absolute value. This shared scaling preserves the relative geometry of the two displayed components. For Qwen-Image, the PCA is computed by singular value decomposition, PC1 is oriented so that its final score is at least its initial score, and PC2 is oriented so that its score at step 25 is nonnegative. For each model,

the three pooled views use exactly the same normalized scores, as do the paired three-dimensional and planar views for each individual prompt. Planar views use equal axis scaling and shared square limits; the Qwen limits are $[-1.18, 1.18]$. Circles mark measured states, blue changes from dark to light with denoising progress, and stars mark the final state (PC1 in the score-versus-step view). Straight segments connect successive measurements without smoothing.

E.3 Pooled Temporal Patterns and Interpretation

FLUX.1-dev. The companion plots in Figure 14 make the timing and geometry of Figure 2 explicit. In B_0, the displayed scores change sharply during the first few steps and remain near their final values over most of the schedule. The middle band B_3 evolves over a broader interval. The high band B_6 changes slowly at first, with larger changes near the end. The planar projections trace the same measured states without perspective, while the score-versus-step curves show when each component evolves.

Qwen-Image. In Figure 15, B_0 also undergoes a sharp initial transient and then changes little in the displayed components. The middle band B_3 evolves over a broader portion of the schedule, while B_6 exhibits an early transient followed by a pronounced change near the end. Across both models, the different timing across bands supports the qualitative motivation of Figure 2 for separate temporal models and band-specific forecasting coordinates.

Each panel has its own PCA basis and normalization, so absolute distances, directions, and amplitudes cannot be compared across models, bands, prompts, or pooled and individual views. A nearly vertical trajectory in the three-dimensional plots indicates little change in PC1 and PC2; variation can still occur in the omitted components. These plots therefore provide qualitative evidence about temporal structure in the retained subspaces, rather than a quantitative measure of full-velocity motion or forecasting error.

E.4 Individual Prompt Trajectories

Figures 16 to 18 report all 12 FLUX.1-dev generations, and Figures 19 to 21 report all 12 Qwen-Image generations, with two prompts per figure. For each prompt, the upper row displays the trajectory with denoising step increasing downward, and the lower row displays the identical normalized PC pairs in the plane. Columns show full velocity, B_0, B_3, and B_6. The selected step labels are $(0, 25, 49)$, $(0, 1, 49)$, $(0, 10, 49)$, and $(0, 40, 49)$, respectively. These separate fits expose variation across generations that pooling can obscure; coordinates remain specific to each model, prompt, and panel.

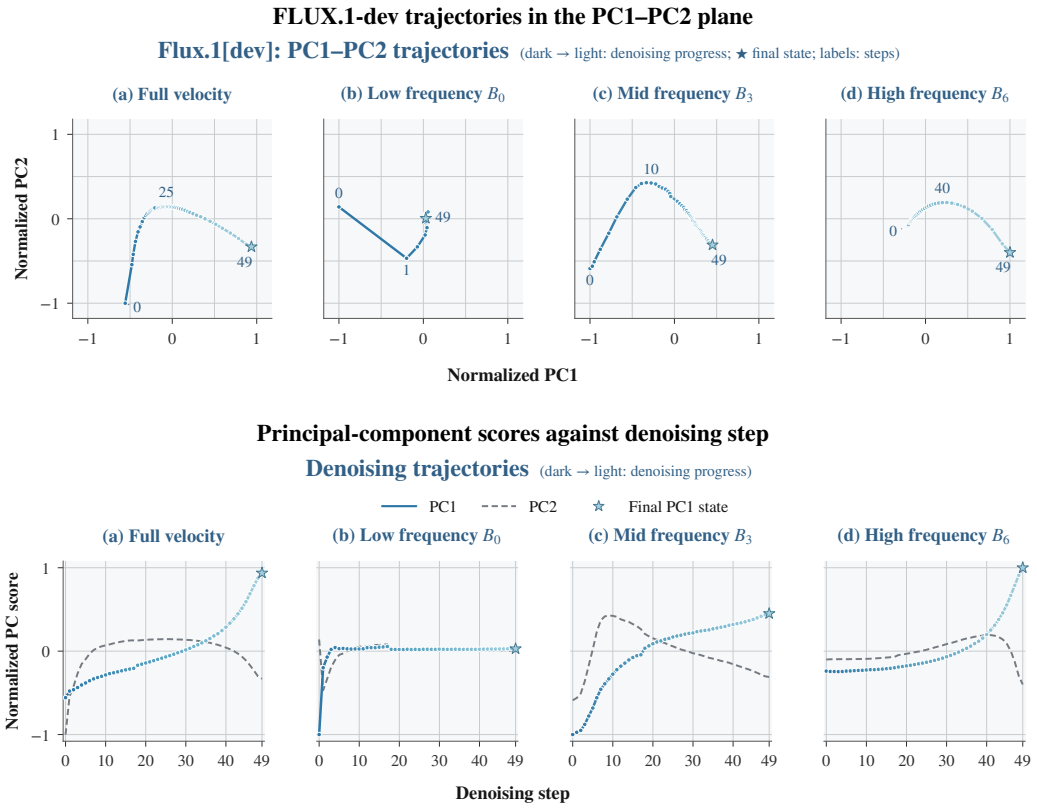

Figure 14: **Complementary views of the pooled FLUX.1-dev trajectories in Figure 2.** Both rows use the same normalized PCA scores as the main figure, pooled across 12 prompts with 50 measured velocities per prompt, guidance scale 3.5, and full transformer computation. Columns show the sampled full velocity and DCT bands B_0, B_3, and B_6. **Top:** PC1–PC2 projections with equal axis scaling; labels identify selected steps. **Bottom:** PC1 (solid blue) and PC2 (dashed gray) against step index. Circles mark measurements, dark to light blue indicates denoising progress, and stars mark the final state (PC1 in the bottom row). Each column has its own PCA basis and scale, constructed as described in Section E.2.

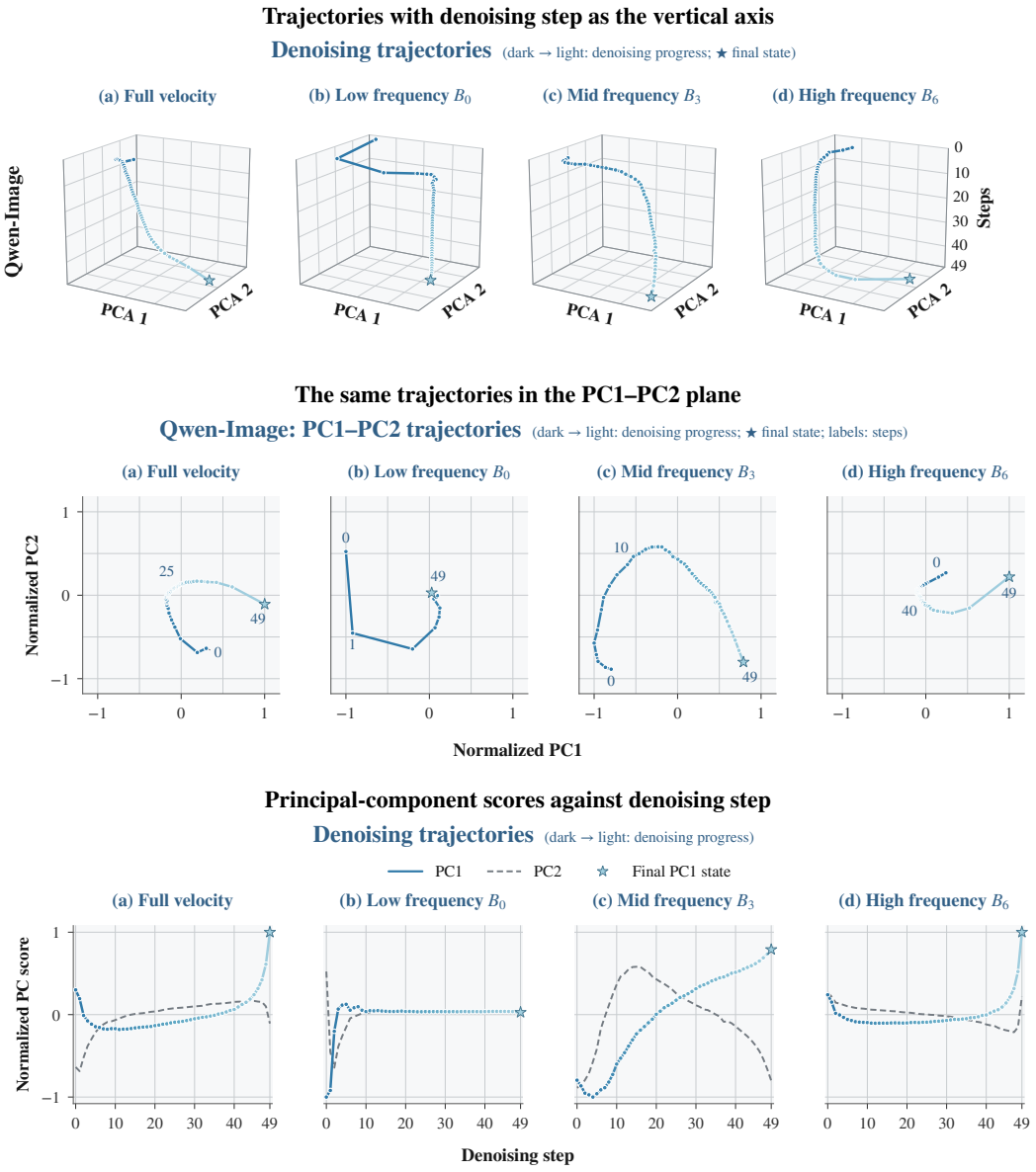

Figure 15: **Spatial-frequency trajectories of Qwen-Image pooled across 12 prompts.** Columns show the sampled full velocity and DCT bands B_0, B_3, and B_6. All rows use the same normalized PCA scores from 50 measured scheduler inputs per prompt. **Top:** Denoising step increases downward. **Middle:** The same PC1–PC2 pairs with equal axis scaling; labels identify selected steps. **Bottom:** PC1 (solid blue) and PC2 (dashed gray) against step index. The low band changes mainly at the start, the middle band evolves over a broader interval, and the high band changes sharply near the end. Bases and scales are fitted independently for each column; see Section E.2 for the construction.

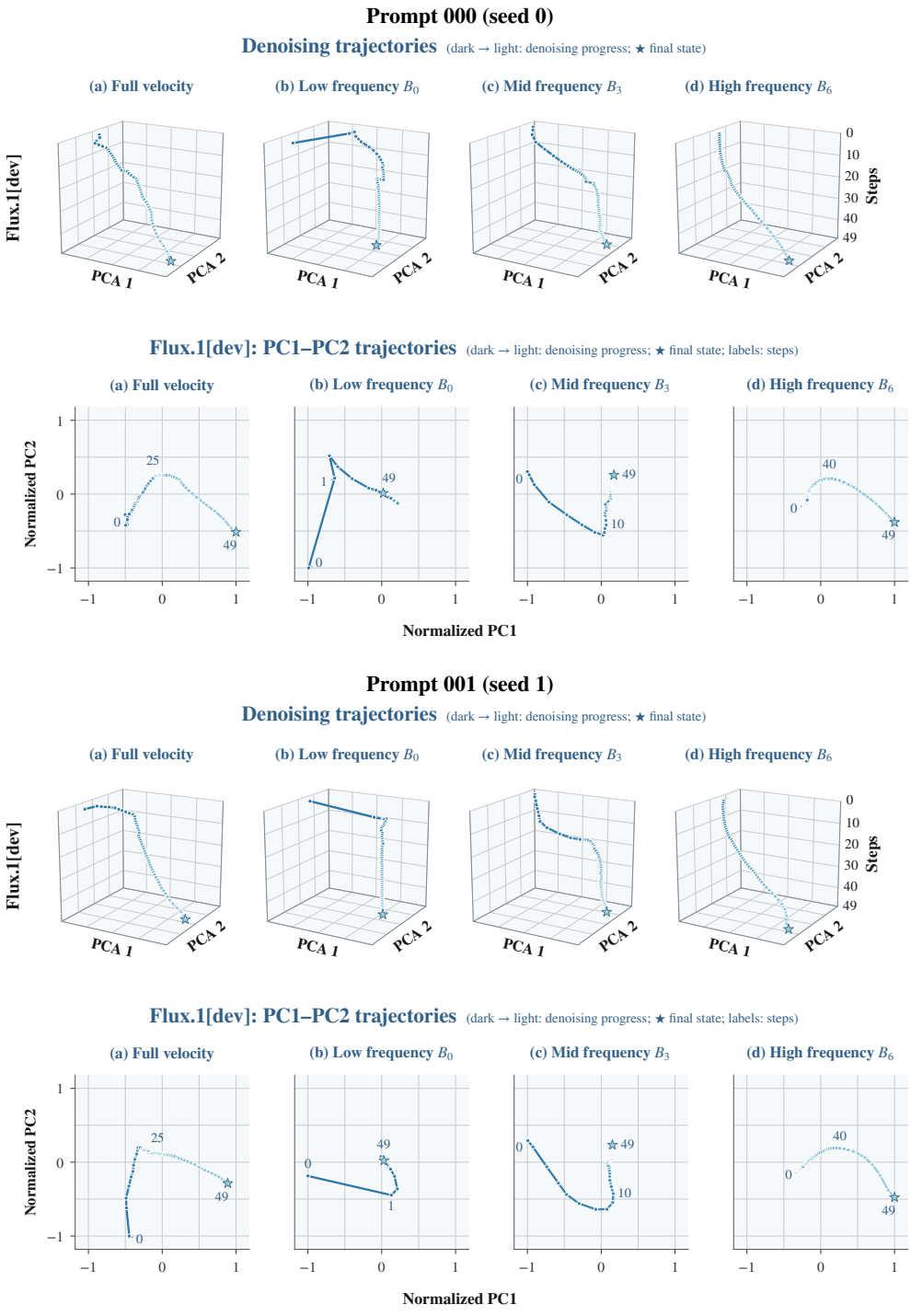

Figure 16: **Individual FLUX.1-dev trajectories: prompts 000 and 001.** Upper pair: "a cat holding a red apple" (seed 0). Lower pair: "a photograph of an astronaut riding a horse on the moon" (seed 1). Each pair shows the same 50 states with an explicit step axis and in the PC1–PC2 plane. Columns show full velocity, B_0, B_3, and B_6; dark to light blue indicates denoising progress and stars mark the final state. PCA bases and scales are specific to each prompt and panel.

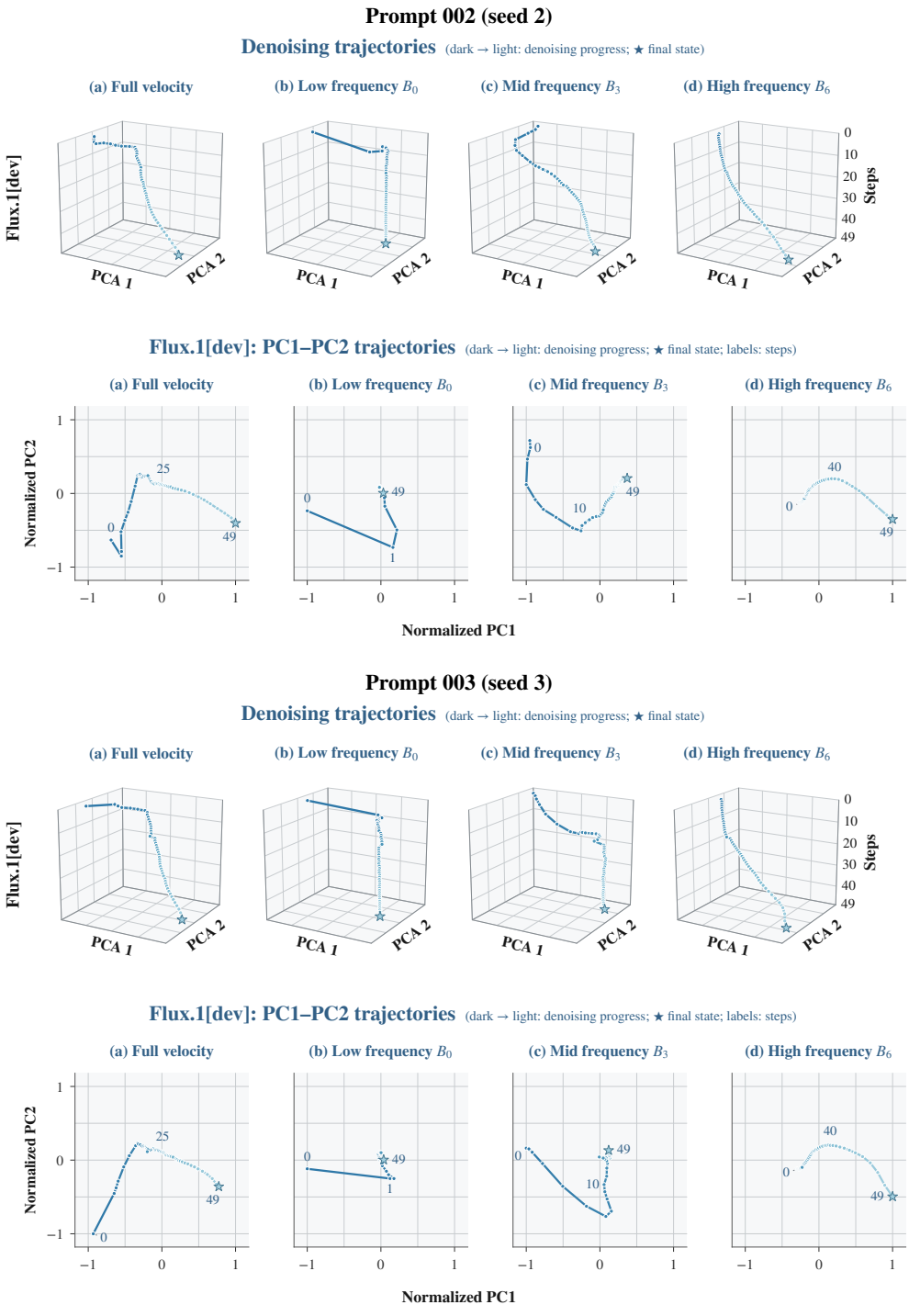

Figure 17: **Individual FLUX.1-dev trajectories: prompts 002 and 003.** Upper pair: "a bowl of ramen with chopsticks, steam rising, cinematic lighting" (seed 2). Lower pair: "a red panda reading a book in a cozy library" (seed 3). Each pair shows the same 50 states with an explicit step axis and in the PC1–PC2 plane. Columns show full velocity, B_0, B_3, and B_6; dark to light blue indicates denoising progress and stars mark the final state. PCA bases and scales are specific to each prompt and panel.

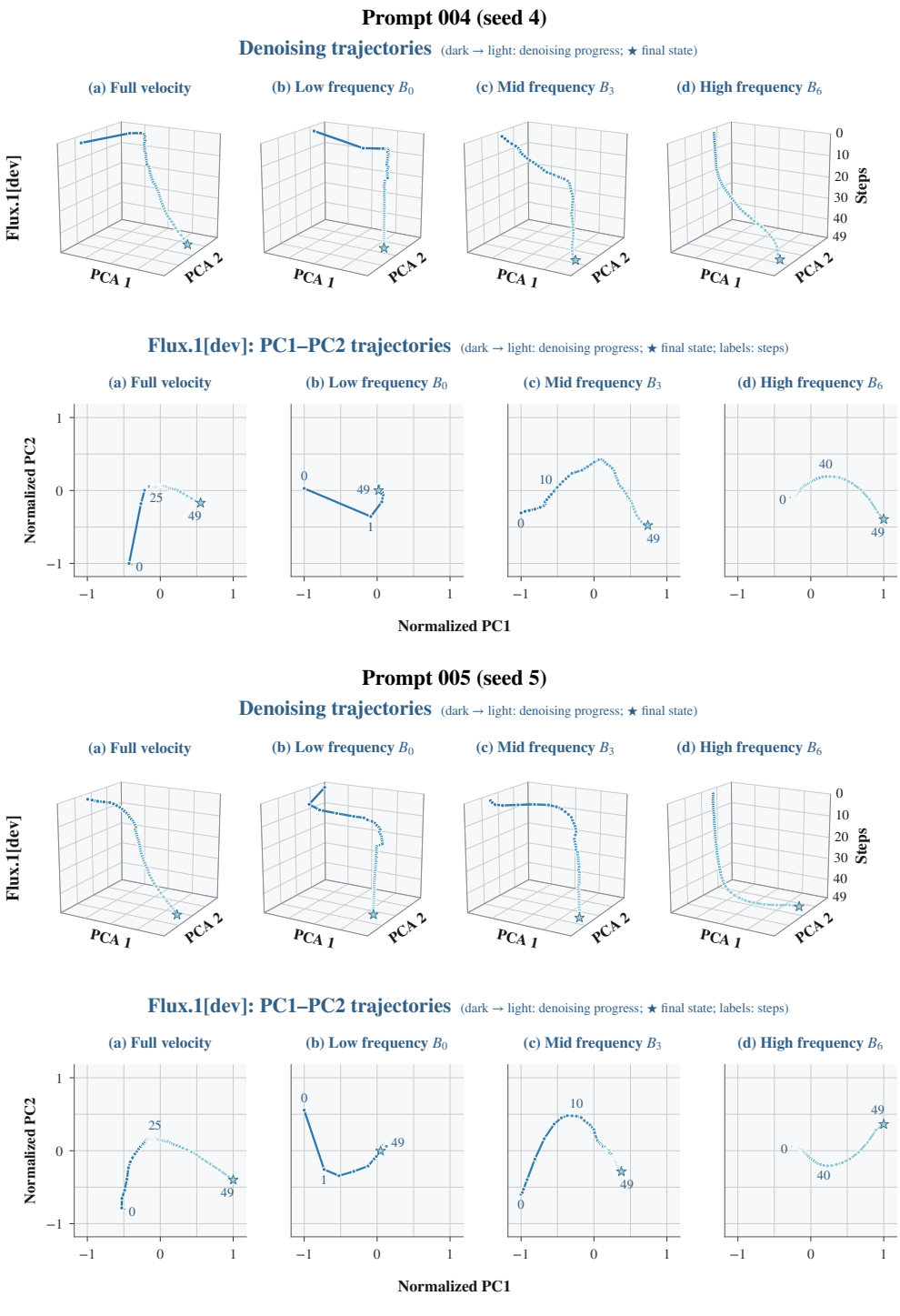

Figure 18: **Individual FLUX.1-dev trajectories: prompts 004 and 005.** Upper pair: "a futuristic city skyline at sunset with flying cars" (seed 4). Lower pair: "a close-up portrait of an old fisherman with a weathered face" (seed 5). Each pair shows the same 50 states with an explicit step axis and in the PC1–PC2 plane. Columns show full velocity, B_0, B_3, and B_6; dark to light blue indicates denoising progress and stars mark the final state. PCA bases and scales are specific to each prompt and panel.

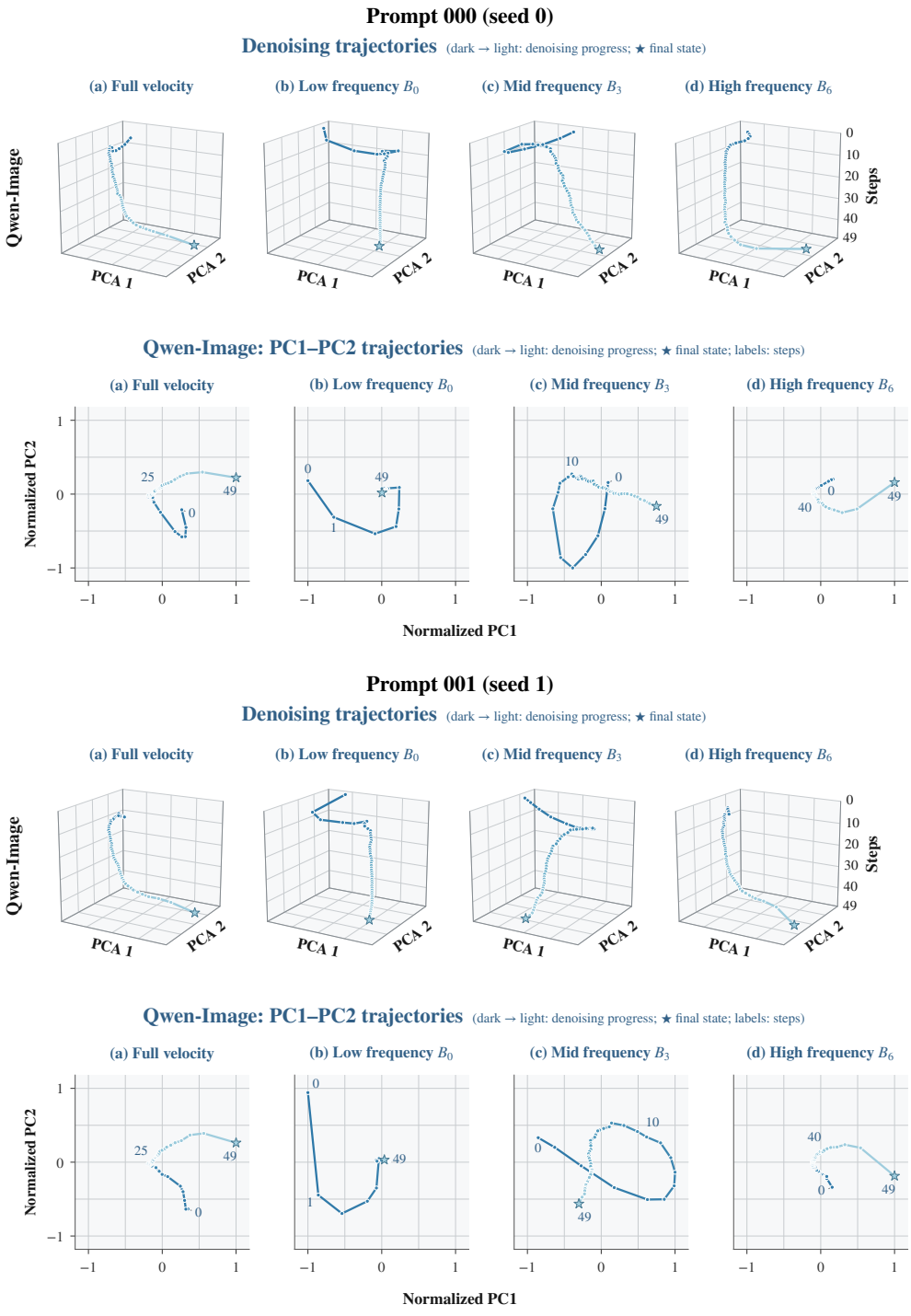

Figure 19: **Individual Qwen-Image trajectories: prompts 000 and 001.** Upper pair: "a cat holding a red apple" (seed 0). Lower pair: "a photograph of an astronaut riding a horse on the moon" (seed 1). Each pair shows the same 50 states with an explicit step axis and in the PC1–PC2 plane. Columns show full velocity, B_0, B_3, and B_6; dark to light blue indicates denoising progress and stars mark the final state. PCA bases and scales are specific to each prompt and panel.

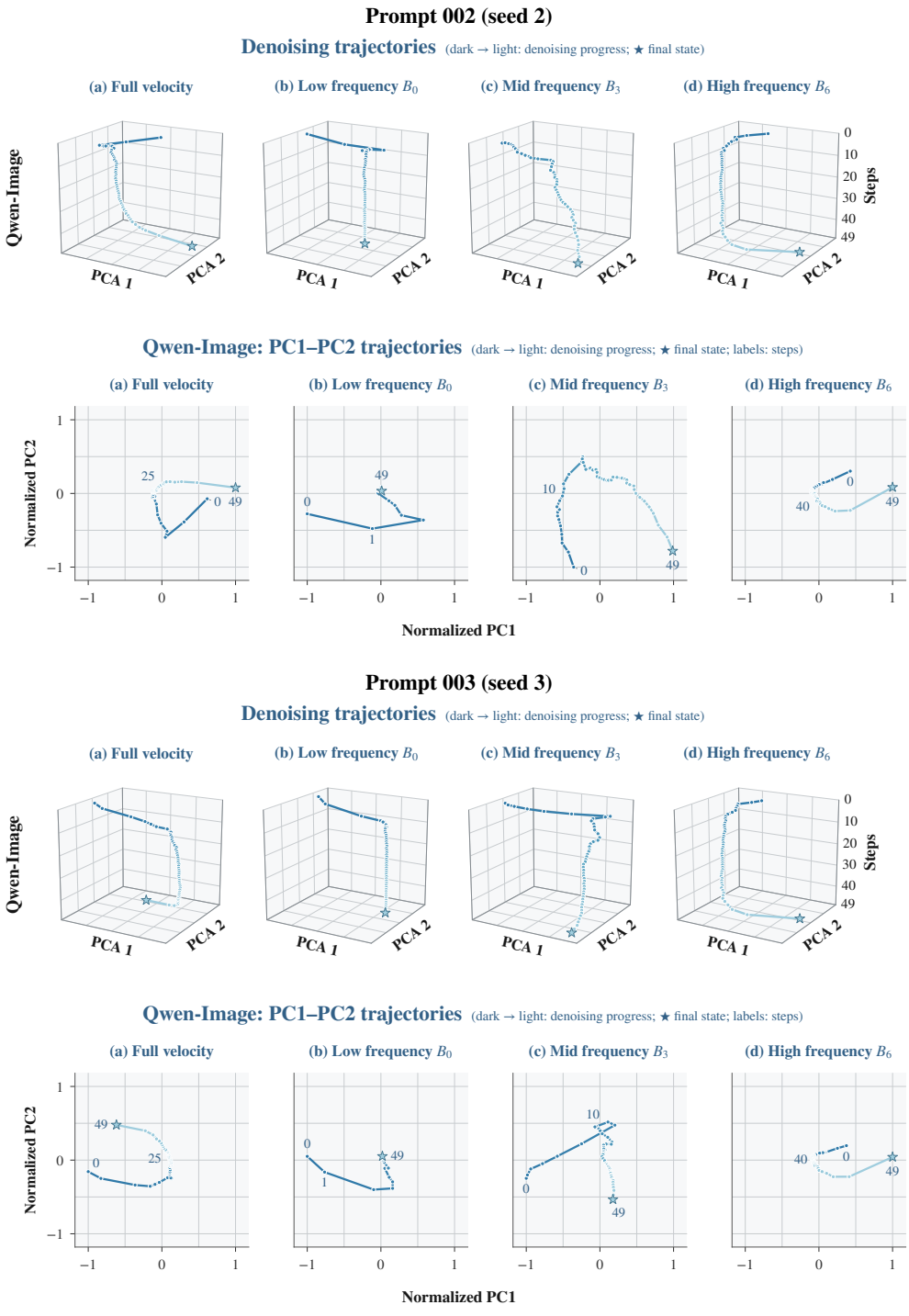

Figure 20: **Individual Qwen-Image trajectories: prompts 002 and 003.** Upper pair: "a bowl of ramen with chopsticks, steam rising, cinematic lighting" (seed 2). Lower pair: "a red panda reading a book in a cozy library" (seed 3). Each pair shows the same 50 states with an explicit step axis and in the PC1–PC2 plane. Columns show full velocity, B_0, B_3, and B_6; dark to light blue indicates denoising progress and stars mark the final state. PCA bases and scales are specific to each prompt and panel.

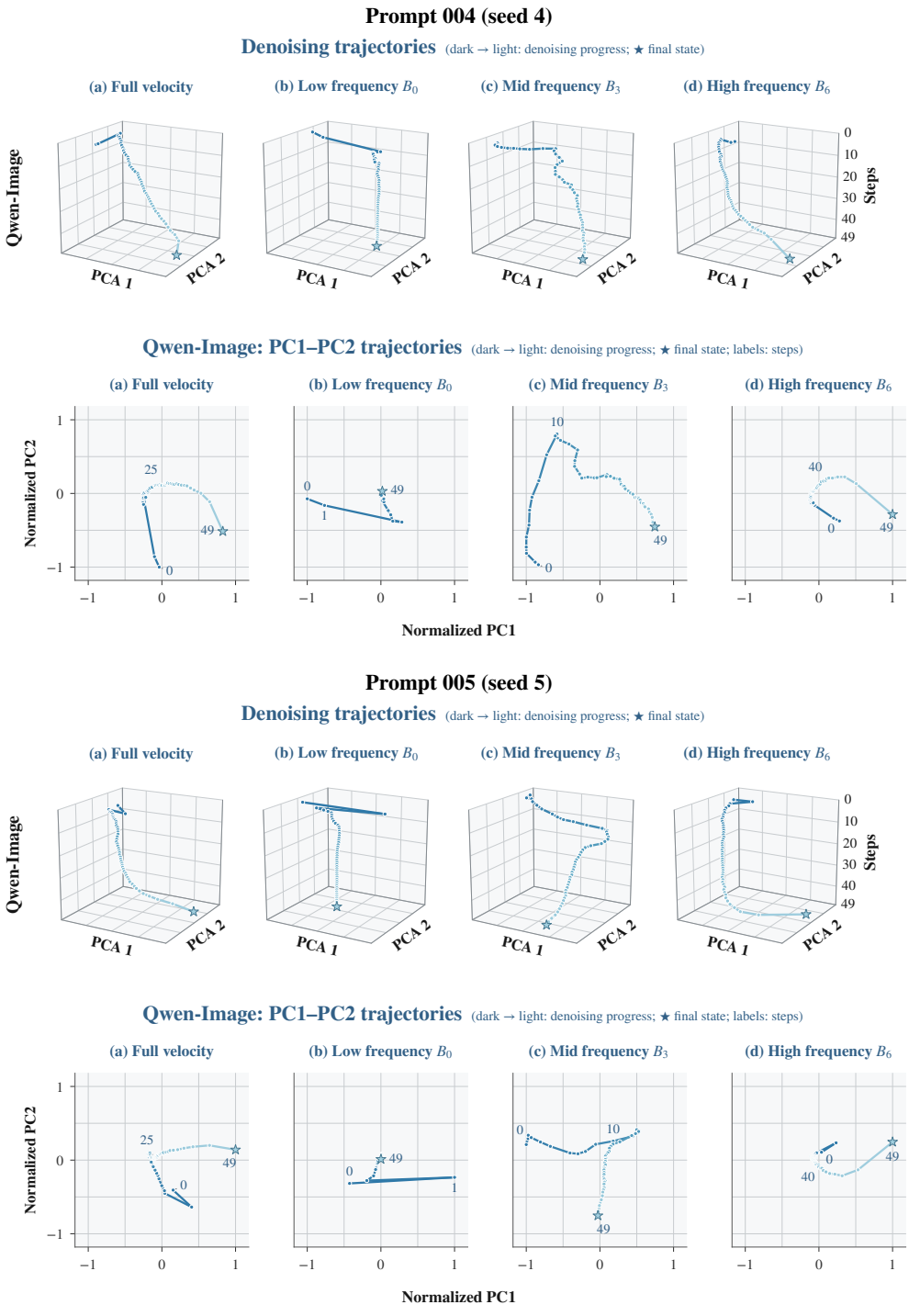

Figure 21: **Individual Qwen-Image trajectories: prompts 004 and 005.** Upper pair: "a futuristic city skyline at sunset with flying cars" (seed 4). Lower pair: "a close-up portrait of an old fisherman with a weathered face" (seed 5). Each pair shows the same 50 states with an explicit step axis and in the PC1–PC2 plane. Columns show full velocity, B_0, B_3, and B_6; dark to light blue indicates denoising progress and stars mark the final state. PCA bases and scales are specific to each prompt and panel.