CUBIC: Curvature-Driven Dynamic Inference Caching for Fast, High-Fidelity Flow Matching

Yuyang Chen[*] **Linqian Zeng**[*] **Yijin Zhou**[†] **Hengjie Li**[‡] **Jidong Zhai**[§]

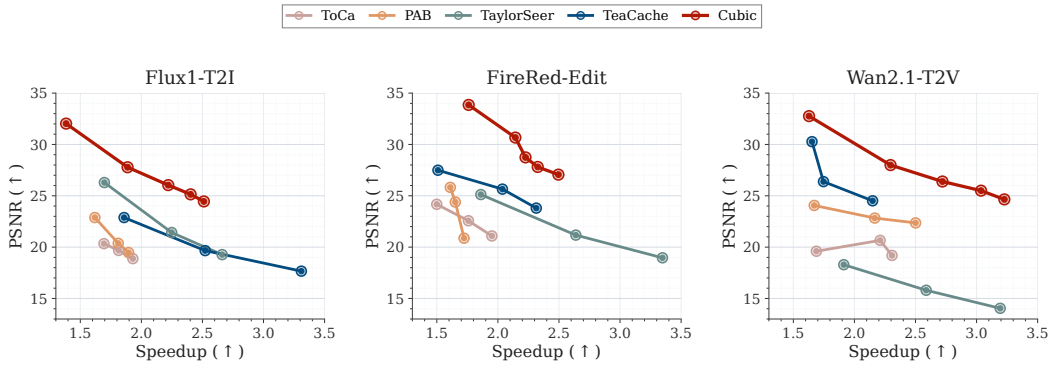

Figure 1: CUBIC consistently achieves Pareto-optimal quality–efficiency trade-offs across all three generation tasks.

Abstract

Flow Matching (FM) has emerged as the dominant paradigm for state-of-the-art image and video generation, yet its reliance on full attention over lengthy sequences incurs significant latency. While existing caching methods reduce redundant computations, they yield suboptimal performance, fundamentally constrained by inflexible spatial granularity and empirical, model-specific heuristics.

To bridge this gap, we ground the caching mechanism in the geometric properties of the FM velocity field. We theoretically establish that the **curvature** inherently dictates the coupling between trajectory fluctuation and computational requirements, thereby providing a principled metric to quantify caching staleness loss and derive update intervals that minimize this loss under a compute budget.

Translating this theoretical insight into practical acceleration, we present CUBIC, a train-free caching framework. It bypasses expensive curvature computations via a lightweight proxy signal that dynamically guides spatial partitioning and budget-constrained update scheduling. At the system level, a sparse forward engine executes these schedules efficiently, turning the reduced token computation into actual FLOP savings and near-linear speedup. Being model-agnostic and controlled by a user-specified acceleration target, CUBIC consistently outperforms state-of-the-art caching baselines in quality at customizable acceleration ratios (averaging 2.5×) across Wan2.1, Flux.1, and FireRed-Image-Edit.[1]

[*]Equal contribution.
[†]Shanghai Jiaotong University. {chen-yy20,028711zlq,708972751}@sjtu.edu.cn
[‡]Shanghai AI Laboratory. lihengjie@pjlab.org.cn
[§]Tsinghua University. zhaijidong@tsinghua.edu.cn
[1]Code is available at https://github.com/chen-yy20/CuBic.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

Flow Matching (FM) [15, 18, 8] has emerged as the dominant paradigm for state-of-the-art image [13, 29], video [28, 12] and multi-modal [27, 33, 3] generation. By learning a neural velocity field to guide the transport path, FM achieves faster convergence and more stable training than traditional diffusion objectives [9, 26]. To capture long-range dependencies across spatiotemporal tokens, these models universally adopt Diffusion Transformers (DiT) [23] equipped with full attention. However, the escalating sequence lengths combined with the quadratic complexity of full-attention impose a prohibitive computational burden. For instance, nearly one NVIDIA A100 hour is required to generate a mere 5-second clip on HunyuanVideo [12], severely hindering real-time and interactive deployment.

Traditional acceleration techniques – such as distillation [22, 25, 14], quantization [37, 35, 38, 36] and sparse kernels [30, 31] — typically require architecture modifications or costly retraining. In contrast, caching-based methods [16, 39, 40, 34, 7, 5] have emerged as a light-weight, train-free alternative. They exploit temporal redundancy across adjacent denoising steps to bypass redundant computations. However, existing caching paradigms are fundamentally hindered by two limitations that prevent them from achieving optimal speed-quality trade-offs:

- **The Granularity Dilemma:** Coarse-grained caching ignores spatial heterogeneity, aggressively skipping computations and causing noticeable artifacts; whereas fine-grained caching introduces prohibitive runtime overhead through fragmented memory access.
- **Reliance on Empirical Tuning:** Most methods depend on model-specific, hand-tuned update intervals that lack a principled optimization objective, with no explicit model of how the chosen intervals affect quality under a fixed compute budget.

To overcome these limitations, we must recognize the true mathematical nature of caching during FM sampling: reusing velocity output for τ steps is equivalent to extending the effective local integration step size of the ODE solver by a factor of τ.

This reframing immediately points to the solution. The "granularity dilemma" is essentially a step-size allocation problem, while the "empirical tuning" issue stems from the lack of a variable to bound numerical error. Both are inherently governed by a single geometric property: the **curvature** of the velocity field. FM training inherently encourages straight trajectories for fast convergence, but deviations from this ideal manifest as high curvature, which requires smaller steps to keep the integration error small. Conversely, low-curvature regions tolerate substantially enlarged steps. Therefore, curvature provides a theoretically grounded indicator for guiding spatial granularity and characterizing staleness error, offering a bridge between empirical caching strategies and principled cache scheduling.

However, translating the theory into practice encounters two immediate barriers (detailed in Sec. 3.2): computing exact curvature requires prohibitive Jacobian operations, and the mathematical mapping from curvature to actual caching error remains undefined.

Taking curvature as the unified theoretical knob, we propose CUBIC, an FM acceleration framework specifically designed to bridge these theoretical barriers into practical speedups.

- **Bypassing the Jacobian Barrier:** CUBIC introduces a lightweight online monitor that estimates curvature via *endpoint prediction drift*, replacing expensive gradient computations with simple algebraic subtraction.
- **Closing the Error Gap:** CUBIC derives a closed-form error bound that scales as $\kappa\tau^2$. Using Lagrangian optimization, it converts this bound into a budget-constrained interval allocation rule, providing a principled alternative to purely empirical threshold tuning.
- **Efficient Execution:** A sparse forward engine translates the proposed caching plans into highly efficient runtime via dynamic KV cache updating, so that the reduced token computation yields actual wall-clock savings while preserving generation quality.

As shown in Fig. 1, experiments across diverse generation tasks—Flux.1 [13] for text-to-image, Wan2.1 [28] for text-to-video, and FireRed [27] for image editing—demonstrate that CUBIC consistently outperforms state-of-the-art caching baselines, delivering superior generation quality at any given computational budget.

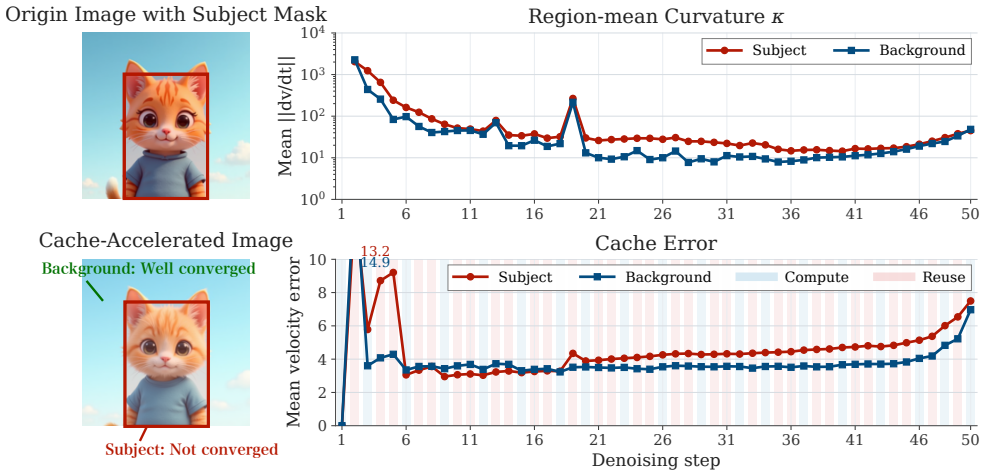

Figure 2: Motivation Experiment: Subject regions exhibits higher curvature and incurs higher caching error accumulation.

2 Related Work

Flow Matching Superseding traditional noise-prediction diffusion models [9, 26], *Flow Matching* (FM) [15, 18] learns a continuous-time velocity field to transport samples from a prior to the data distribution along simple probability paths. A canonical example is the straight-line interpolation path:

$$x_t = (1-t)\,x_0 + t\,x_1, \quad t \in [0,1]$$

This formulation yields a smoother, mathematically tractable vector field, drastically reducing the ODE integration steps required for high-fidelity generation. FM now serves as the default backbone for state-of-the-art generative architectures [24, 13, 28, 12, 6, 27].

Cache-based Acceleration Cache-based methods offer a training-free, plug-and-play acceleration paradigm by reusing redundant intermediate features along the denoising trajectory. Existing works explore a vast spectrum of caching granularities: from coarse step-level [16, 21, 7, 17] or layer-level skipping [34, 1], to fine-grained attention partial reuse [39, 4, 32], and down to the most granular token-level selective updates [40, 19, 5]. However, navigating this design space is fundamentally hindered by the granularity dilemma and an over-reliance on empirical heuristics.

Optimized Samplers and Kernels Another paradigm focuses on modifying the solving process or computational operators. Algorithmically, distillation and reflow techniques [22, 18] reduce integration steps by distilling generation trajectories into simpler paths. System-wise, quantized [37] or sparse attention kernels [30] are developed to mitigate the quadratic attention bottleneck in long sequences. Despite their effectiveness, these methods typically require auxiliary training data or architecture-specific modifications. Importantly, they are orthogonal to CUBIC, allowing for potential cumulative speedups.

3 Motivation

3.1 Observation: Spatially Heterogeneous Computational Demands

During inference, Flow Matching models exhibit significant spatial heterogeneity in computational demands [5]. Specifically, different spatial regions within a generated sample require vastly different amounts of computation to reach convergence. Usually, flat and static backgrounds converge faster than complex and dynamic subjects. This intrinsic non-uniformity implies immense potential for content-aware selective computation. However, to truly harness this characteristic, a fundamental question remains: *what intrinsic property of the generation dynamics dictates these varying computational requirements?*

Fig. 2 provides an empirical clue. Compared with the background, the subject region shows higher trajectory curvature and accumulates larger cache-induced velocity errors, indicating that trajectory curvature is closely tied to region-wise cache tolerance.

3.2 Theoretical Essence: Integration Step Size and Trajectory Curvature

To understand why the cache-induced error in Fig. 2 varies across regions, we seek the answer from the Ordinary Differential Equation (ODE) perspective of Flow Matching. During inference, the continuous-time ODE is solved using a discrete time grid $0 = t_0 < t_1 < \cdots < t_K = 1$. Denoting the step size as $\Delta t = t_k - t_{k-1}$ (which can be uniform or non-uniform), the standard forward Euler iteration is formulated as:

$$x_{k+1} = x_k + \Delta t \cdot v_\theta(x_k, t_k). \tag{1}$$

As illustrated in Fig. 3, cache reuse can be interpreted as holding the velocity prediction fixed over an extended interval. In other words, reusing cached velocity features for multiple steps effectively enlarges the local integration step size. We therefore conceptually reframe step requirements across different regions as their *permissible integration step sizes*: fewer required updates imply that a region can tolerate a larger effective Δt.

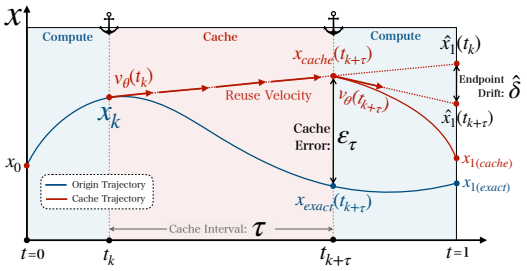

Figure 3: Geometric interpretation of velocity reuse, cache error and endpoint drift in FM velocity field.

To achieve fast sampling, the FM training objective [18, 8] inherently encourages the learned velocity field v_θ to be as constant as possible along the temporal dimension, guiding the generation trajectory towards a straight line, i.e., $dv_\theta/dt \equiv 0$. If the trajectory were perfectly straight, a single large Euler step would solve the ODE exactly. In practice, however, trajectories deviate from this ideal, and such deviation determines whether enlarging effective step size is reliable.

We quantify this deviation as the **curvature** (κ) of the trajectory, mathematically captured by the material derivative of the velocity field along the flow:

$$\kappa(t) = \frac{\partial v_\theta}{\partial t} + (\nabla_x v_\theta) \cdot v_\theta. \tag{2}$$

High-curvature trajectories—driven by drastic velocity fluctuations—are highly sensitive to step size; aggressively enlarging the step introduces large truncation errors, so these regions require finer integration. Conversely, low-curvature trajectories exhibit highly consistent velocities, where a single large step (more caching steps) is nearly equivalent to multiple smaller ones.

3.3 From Theory to Practice: The Practical Barriers

While the theory suggests a curvature-aware caching strategy, realizing it in practice encounters two immediate barriers. First, exact curvature is computationally prohibitive. Obtaining κ via Eq. 2 requires computing the spatial Jacobian $\nabla_x v_\theta$ for massive sequences, introducing an overhead that defeats the purpose of caching. Second, the theoretical gap remains open. Even with a free oracle for κ, the mapping from ODE trajectory curvature to the actual feature-reuse "staleness error" has not been established. Without a closed-form error bound, a curvature-based design still has to tune empirically how curvature translates into update intervals.

4 Methodology

We present CUBIC, a curvature-driven dynamic caching framework for FM generation. The core idea is to connect local trajectory geometry with runtime caching decisions: regions whose trajectories are locally straighter can tolerate more aggressive cache reuse, whereas regions with higher curvature should be refreshed more frequently.

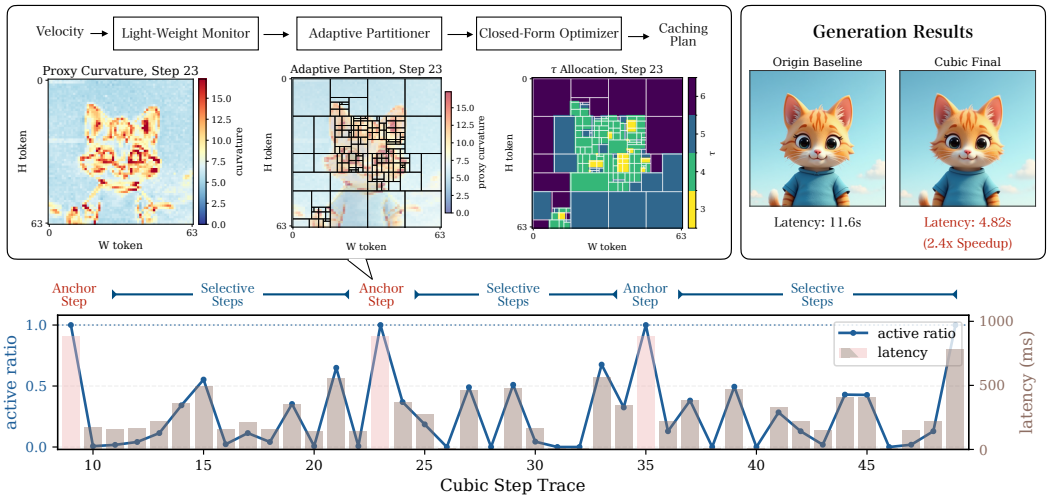

Figure 4: CUBIC overview.

4.1 Overview of the Inference Pipeline

As illustrated in Fig. 4, CUBIC exploits the spatial heterogeneity of permissible step sizes by transforming the standard ODE solving process into an *anchor-selective inference loop*.

The pipeline alternates between two types of steps:

- **Anchor Steps:** Full forward passes that act as the "brain" of the system. They refresh the exact velocity prediction for all tokens, measure the geometric properties of the trajectory, and dynamically re-evaluate the caching policy.
- **Selective Steps:** Sparse forward passes that act as the "muscle" for acceleration. They compute only a dynamically chosen subset of tokens (active tokens), while the remaining tokens (frozen tokens) reuse cached intermediate states.

This anchor-selective decoupling divides the acceleration mechanism into two clean phases: *estimation & optimization* at anchor steps (Sec. 4.2), and *sparse execution* at selective steps (Sec. 4.3). The two phases rest on different kinds of justification. The curvature proxy, the error model, and the budget-constrained interval allocation are derived from the ODE analysis in Appendix A. The anchor horizon and the block partition are empirical system choices: they do not change the allocation rule, but select an operating point on the latency–quality frontier for a given architecture (Appendix E).

4.2 Anchor Steps: Estimation and Optimization

Curvature Estimation Directly computing local trajectory curvature is impractical at inference time because it requires Jacobian information which is expensive to compute. We therefore seek an online proxy that introduces light weight system cost. Our key observation is that if a token trajectory is locally straight, then its endpoint predicted by the current velocity field should remain nearly unchanged across successive observations. In contrast, trajectory curvature causes this endpoint prediction to drift.

For a spatial token indexed by i, we define its endpoint prediction at time t as

$$\hat{x}_1^{(i)}(t) = x^{(i)}(t) + (1-t)\, v_\theta^{(i)}(x(t), t). \tag{3}$$

Importantly, the monitor is updated only at anchor steps rather than at every sampler step. Let $\{t_{a_m}\}_{m=1}^{M}$ denote the anchor-step sequence. Interpreted in Fig. 3, we define the endpoint prediction drift between two consecutive anchor observations as

$$\hat{\delta}^{(i,m)} = \hat{x}_1^{(i)}(t_{a_m}) - \hat{x}_1^{(i)}(t_{a_{m-1}}). \tag{4}$$

As derived in Appendix A.1, to first order this drift scales proportionally to the local trajectory curvature $\kappa^{(i)}(t_{a_m})$, modulated by the remaining time horizon $(1 - t_{a_m})$ and the step interval $\Delta t_{a_m} = |t_{a_m} - t_{a_{m-1}}|$.

Rearranging this relation directly gives the online curvature estimate:

$$\hat{\kappa}^{(i)} = \left\| \hat{\delta}^{(i,m)} \right\| / \left[(1 - t_{a_m}) \, \Delta t_{a_m} \right]. \tag{5}$$

In practice, we compute this proxy in latent space and aggregate it to the token grid using the same patch granularity as the underlying DiT tokenizer, producing a token-space curvature map $\kappa \in \mathbb{R}^{F_t \times H_t \times W_t}$.

However, assigning independent caching intervals to individual tokens incurs prohibitive scheduling overhead. Instead, CUBIC groups adjacent tokens into rectangular blocks $\{B_b\}$ that share a unified update interval. Sharing an interval also keeps neighboring tokens, which attend to each other with high attention weights in DiT self-attention, at the same cache age. Under per-token schedules, adjacent tokens would read keys and values computed at different steps (Sec. 4.3), and this local inconsistency propagates through subsequent layers. To balance spatial granularity and scheduling efficiency, we employ a lightweight partitioning strategy guided by the curvature map (detailed in Appendix B). Once the partition $\mathcal{P} = \{B_b\}$ is established, we aggregate the token-wise proxy into a representative block-wise score using a spatial mean: $\kappa_b = \frac{1}{|B_b|} \sum_{i \in B_b} \hat{\kappa}^{(i)}$. This block-wise curvature map $\{\kappa_b\}$ serves as the direct input for the subsequent budget-constrained interval optimization.

Cache Error Modeling After obtaining a representative curvature for each block, CUBIC determines how long each block may safely reuse cached outputs. Consider a block b that is fully recomputed at an anchor step t_0, yielding the cached velocity $v^{(b)}_{\text{cache}} = v^{(b)}_{\theta}(x(t_0), t_0)$. If this block is frozen for the subsequent τ_b selective steps, the same cached velocity is continuously reused throughout the interval $[t_0, t_0 + \tau_b \Delta t]$.

Let $x^{(b)}_{\text{exact}}(t)$ denote the exact trajectory, and $x^{(b)}_{\text{cache}}(t)$ denote the trajectory obtained by reusing the cached velocity. We define the cache-induced state truncation error at the end of the reuse window as:

$$\varepsilon^{(b)}_{\tau_b} = x^{(b)}_{\text{cache}}(t_0 + \tau_b \Delta t) - x^{(b)}_{\text{exact}}(t_0 + \tau_b \Delta t). \tag{6}$$

As derived in Appendix A.2, if the curvature is bounded within the reuse window, this accumulated error is bounded by the local trajectory curvature:

$$\left\| \varepsilon^{(b)}_{\tau_b} \right\| \lesssim \frac{1}{2} \, \kappa_b \, \tau_b^2 \, \Delta t^2. \tag{7}$$

This result directly exposes the desired behavior: blocks with low curvature can tolerate longer reuse intervals, while high-curvature blocks should be updated more often.

Update Interval Optimization We formalize interval allocation as a budget-constrained optimization problem. Let ρ denote the target speedup, τ_b the update interval for block b, and $|\mathcal{B}_b|$ its spatial size (i.e., number of tokens). Because the adaptive partitioner yields blocks of varying sizes, the optimization must be weighted by block size.

$$\min_{\tau_b \geq 1} \sum_{b=1}^{\mathcal{B}} |\mathcal{B}_b| \, \kappa_b^{\alpha} \tau_b^{\beta} \quad \text{s.t.} \quad \sum_{b=1}^{\mathcal{B}} \frac{|\mathcal{B}_b|}{N} \frac{1}{\tau_b} = \frac{1}{\rho}, \tag{8}$$

where $N = \sum_{b=1}^{\mathcal{B}} |\mathcal{B}_b|$ is the total number of tokens in the image, $\rho > 1$ is the target speedup factor, and $\tau_b \geq 1$ are the block-wise update intervals. The objective function minimizes the total expected squared error across the entire image, weighted by the number of tokens $|\mathcal{B}_b|$ per block, with the error bound per block squaring to $\kappa_b^{\alpha} \tau_b^{\beta}$. The constraint ensures that the token-weighted average update rate $\mathbb{E}[1/\tau_b]$ equals the reciprocal of the target speedup $1/\rho$, exactly enforcing the compute budget. Note that ρ specifies the compute budget of the selective steps only. Anchor steps and the full-compute steps at the beginning and end of sampling process all tokens. With K_{full} full-compute steps and K_{sel} selective steps, the overall token-compute fraction is $(K_{\text{full}} + K_{\text{sel}}/\rho)/(K_{\text{full}} + K_{\text{sel}})$, and the measured wall-clock speedup additionally reflects system overhead. Intuitively, this ensures proportional FLOP savings: freezing a large block (high $|\mathcal{B}_b|$) for τ_b steps yields strictly greater computational reduction than freezing a smaller block with the same interval.

As derived in Appendix A.3 via the Lagrange multiplier method, the block size weights $|\mathcal{B}_b|$ appear on both the objective and the constraint. Consequently, they cancel out during the first-order optimality condition. The resulting optimum is $\tau_b^\star \propto \kappa_b^{-\gamma}$ with $\gamma = \alpha/(\beta+1)$, so the allocation depends on the two error exponents only through γ. Substituting the theoretical defaults $(\alpha, \beta) = (2, 4)$ gives $\gamma = 2/5$ and yields the final allocation rule:

$$\tau_b^\star \propto \kappa_b^{-2/5}. \tag{9}$$

Thus, CUBIC naturally assigns shorter intervals to high-curvature blocks and longer intervals to low-curvature ones, irrespective of their spatial sizes. In implementation, the continuous solution is discretized to valid integer bounds.

Effective ODE Strides To ensure optimized intervals translate into actual FLOP savings, anchor steps dictate a selective phase length of $2\tau_{\max}$, where $\tau_{\max} = \max_{b\in\mathcal{P}} \tau_b$. This receding-horizon rule is tied to the current allocation: it gives the blocks assigned the longest interval enough steps to execute their reuse schedule before curvature is re-estimated, so that caching operations serve as *effective ODE strides* rather than being overwritten by the next anchor step. The factor of two is an empirical default rather than a theoretical optimum. If the curvature of a block changes within the window, its schedule is stale only until the next anchor step, which re-measures curvature and assigns shorter intervals to blocks whose curvature has increased. Appendix E.3 compares this rule with shorter and longer horizons and with a fixed anchor gap.

4.3 Selective Steps: Sparse DiT Execution

Sparse Forward with KV Cache Given the partition $\mathcal{P}$ and the block intervals $\{\tau_b\}$ at anchor step, the active blocks at selective step k are defined as

$$\mathcal{A}_k = \{b \in \mathcal{P} \mid s_k \bmod \tau_b = 0\}, \tag{10}$$

where s_k is the local step index within the selective phase. The interval τ_b is an integer with $\tau_b \geq 1$. A block with $\tau_b = 1$ is recomputed at every selective step, and a block with $\tau_b > 1$ is recomputed every τ_b steps and stays frozen in between. This update-and-reuse cycle repeats until the next anchor step. The active blocks induce the active and frozen token sets

$$\Omega_k^{\text{act}} = \bigcup_{b\in\mathcal{A}_k} \mathcal{B}_b, \qquad \Omega_k^{\text{frz}} = \Omega \setminus \Omega_k^{\text{act}}. \tag{11}$$

At each DiT layer, CUBIC only computes the active tokens. To maintain full-sequence spatial dependencies without recomputing frozen tokens, the attention mechanism for active tokens queries against a reconstructed full sequence. Specifically, the newly computed Key/Value (KV) tensors for active tokens are concatenated with the cached KV tensors of frozen tokens to form the full KV states:

$$K_\ell^{\text{full}} = K_\ell^{\text{act}} \oplus K_\ell^{\text{frz}}, \quad V_\ell^{\text{full}} = V_\ell^{\text{act}} \oplus V_\ell^{\text{frz}}. \tag{12}$$

Attention is then evaluated as:

$$Y_\ell^{\text{act}} = \text{Attn}\big(Q_\ell^{\text{act}}, K_\ell^{\text{full}}, V_\ell^{\text{full}}\big). \tag{13}$$

Crucially, after the attention computation, CUBIC updates the cached KV entries for the active tokens in-place with the newly computed K_ℓ^{act} and V_ℓ^{act}. This ensures that the per-layer KV cache progressively incorporates the latest cross-token interactions across successive steps, rather than remaining strictly static. The final output of the selective step is then formed by merging newly computed active-token outputs with cached frozen-token outputs.

This selective forward engine is the component that actually yields computational savings: active tokens traverse the full DiT computation path, whereas frozen tokens participate only through cache reuse.

5 Evaluation

5.1 Experiments Setup

Models and Baselines. We evaluate CUBIC across three representative generative settings: text-to-image generation using Flux-1-dev [13], instruction-guided image editing using FireRed-Image-Edit-

Table 1: Quantitative evaluation of inference efficiency and visual quality in generation models.

Model	Method	Latency(s) ↓	Speedup ↑	PSNR ↑	SSIM ↑	LPIPS ↓	FID/FVD ↓	HPSv3/VBench ↑
Flux-1	Baseline	11.59 ± 0.05	–	–	–	–	–	10.68
	PAB	6.40 ± 0.11	1.81×	20.37	0.7702	0.2574	77.37	10.65
	TeaCache	6.24 ± 0.08	1.86×	22.88	0.8514	0.1490	45.87	10.83
	ToCa	6.39 ± 0.04	1.81×	19.68	0.7493	0.2726	70.76	10.62
	TaylorSeer	6.83 ± 0.08	1.70×	26.29	0.8920	0.0994	**26.80**	10.82
	CUBIC (std)	6.14 ± 0.24	1.89×	**27.78**	**0.8998**	**0.0946**	27.81	10.87
	CUBIC (fast)	$\mathbf{4.82 \pm 0.21}$	**2.40**×	25.14	0.8457	0.1730	47.32	**10.97**
FireRed	Baseline	39.14 ± 0.09	–	–	–	–	–	8.17
	PAB	23.70 ± 0.17	1.65×	24.39	0.8888	0.0929	39.41	7.95
	TeaCache	16.91 ± 0.26	2.31×	23.80	0.8955	0.0995	41.03	7.83
	ToCa	20.06 ± 0.04	1.95×	21.09	0.7884	0.1779	67.25	7.18
	TaylorSeer	21.04 ± 0.04	1.86×	25.12	0.9102	0.0837	33.70	**8.07**
	CUBIC (std)	18.26 ± 0.57	2.14×	**30.67**	**0.9538**	**0.0468**	**18.55**	8.00
	CUBIC (fast)	$\mathbf{16.83 \pm 0.42}$	**2.33**×	27.81	0.9201	0.0907	32.18	7.90
Wan 2.1	Baseline	101.54 ± 2.40	–	–	–	–	–	79.90
	PAB	40.60 ± 0.10	2.50×	22.36	0.8000	0.1670	57.78	78.41
	TeaCache	61.44 ± 1.92	1.65×	30.27	0.9270	0.0454	11.72	79.79
	ToCa	44.00 ± 2.26	2.31×	19.19	0.7383	0.2405	84.04	78.42
	TaylorSeer	39.24 ± 2.56	2.59×	15.80	0.6410	0.3521	137.06	78.82
	CUBIC (std)	62.35 ± 2.77	1.63×	**32.76**	**0.9481**	**0.0317**	**7.92**	79.37
	CUBIC (fast)	$\mathbf{37.32 \pm 0.62}$	**2.72**×	26.39	0.8740	0.0986	31.76	**80.03**

1.0 [27], and text-to-video generation using Wan2.1-T2V-1.3B [28]. We compare with state-of-the-art feature caching baselines, including TokenCache (ToCa) [40], TeaCache [16], Pyramid Attention Broadcast (PAB) [39], and TaylorSeer [17].

Metrics and Evaluation Datasets We report end-to-end latency and evaluate quality with PSNR, SSIM, LPIPS, and FID/FVD, plus HPSv3 [20] for images and VBench metrics for videos, on t2i-diversity-eval [2], PIE-Bench++ [10], and VBench [11], respectively. PSNR, SSIM, and LPIPS are measured against the uncached model under identical inputs (Appendix D).

Implementation Experiments run on NVIDIA H100 using PyTorch. CUBIC acts as a non-intrusive wrapper, and all methods share identical prompts and seeds. Baselines use their official implementations on Flux and Wan; for FireRed, which lacks official support, we adapt them from analogous models (Appendix D). Code is available at `https://github.com/chen-yy20/CuBic`.

5.2 Main Results: Pareto-Optimal Acceleration

Quantitative Comparison We evaluate the quality and efficiency of images and videos generated with CUBIC under a standard and a fast setting (Table 3) against baseline methods, and summarize the results in Table 1. Compared with existing caching-based acceleration methods, CUBIC achieves higher speedups while preserving better generation quality. Notably, for models where attention accounts for a larger fraction of the overall computation, such as Wan2.1, CUBIC can remove redundant computation more effectively, leading to more pronounced acceleration gains.

Sensitivity Test on Quality-Latency Trade-off Fig. 1 reports the PSNR–speedup trade-off of CUBIC and the compared baselines across different models. Across a wide range of speedup ratios, CUBIC consistently attains higher PSNR (with correspondingly stronger SSIM, LPIPS, and FID/FVD; see Appendix D), thereby lying on the Pareto frontier of the quality–latency trade-off. Against the token-level method Jano [5], CUBIC also achieves higher fidelity at comparable or higher speedup on both Flux and Wan2.1 under a held-out prompt protocol (Appendix E.1).

5.3 Validating Theoretical Foundations

Accuracy of Curvature Estimation We first validate whether the proposed endpoint-drift proxy accurately reflects true trajectory curvature. As shown in Fig. 5(a-b), the proxy closely matches both the spatial distribution and the relative magnitudes of the true curvature, exhibiting a strong linear correlation. This confirms that the proxy serves as a highly reliable online approximation for $\kappa^{(i)}$.

Verification of the Cache Error Modeling We validate the cache-error bound derived in Eq. 7. Fig. 5(c-d) confirm that the cache-induced error scales approximately quadratically with the reuse in-

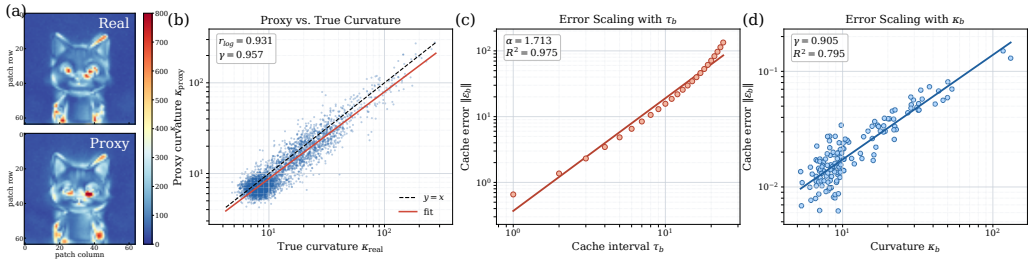

Figure 5: Validating the theoretical foundations of CUBIC. (a) Visualization of real and proxy trajectory curvature on Flux. (b) Patch-wise correlation between κ_{proxy} and κ_{real}. (c) Error scaling with reuse interval τ. (d) Error scaling with curvature proxy κ_b.

terval τ and linearly with the curvature proxy κ. Fitting the squared error gives $(\alpha, \beta) = (1.81, 3.43)$, close to but not identical with the theoretical $(2, 4)$. Since the interval allocation depends on the exponents only through $\gamma = \alpha/(\beta + 1)$, the relevant comparison is $\gamma = 0.409$ (fitted) vs. 0.400 (theoretical). Under a fixed compute budget, the two values produce nearly identical discrete schedules and the same PSNR, SSIM, and LPIPS within display precision (Appendix E.2). The error model therefore captures the dominant scaling of cache-induced degradation; we do not claim that $\gamma = 2/5$ is the empirical optimum.

5.4 Ablation Studies on Core Components

In Table. 2, we measure overhead and quality impact of CUBIC components on Flux with standard setup.

Monitor: Lightweight Proxy Compared with the exact curvature computed via automatic differentiation, our drift proxy reduces the monitoring overhead by more than two orders of magnitude while keeping SSIM nearly unchanged (0.964 vs. 0.967), at the cost of a moderate FID increase (37.5 vs. 33.1). The proxy thus retains most of the scheduling signal without computing explicit Jacobians.

Partitioner: Spatial Locality Per-token scheduling incurs prohibitive indexing overhead and, despite its finer granularity, lower fidelity: it lets adjacent tokens, which attend to each other with high weights, carry keys and values of different cache ages. Block-wise scheduling keeps neighbors on the same schedule, reducing scheduling cost and improving FID and SSIM. The preferred block granularity is model-dependent (Appendix E.4).

Table 2: Ablation studies. Over.: overhead (ms).

Variant	Over.	FID↓	SSIM↑
Monitor			
Exact Jacobian	603.9	33.1	0.967
Proxy	**1.64**	37.5	0.964
Partitioner			
Per-Token	123.9	57.4	0.952
Block-wise	**6.97**	**37.5**	**0.964**
Sparse Executor			
w/o KV	1.83	87.5	0.923
Static KV	2.08	74.7	0.932
Dynamic KV	2.12	**37.6**	**0.964**

Sparse Executor: Dynamic KV and Linear Scaling While a static KV cache prevents contextual isolation, it inevitably accumulates staleness errors. Our dynamic in-place update mechanism reduces this error by continuously refreshing the KV pool, minimizing staleness at negligible I/O cost. Notably, layer-wise KV caches introduce memory requirements that scales with sequence length and layer count (e.g., 0.35GB for Flux vs. 17.68GB for Wan). To ensure deployability on memory-constrained hardware, we integrate asynchronous CPU offloading, which completely eliminates the GPU memory footprint of the cache at a marginal 5% latency overhead.

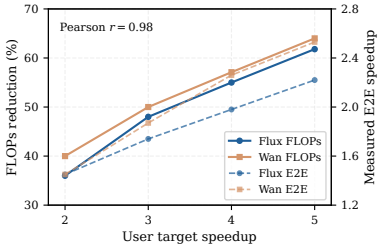

Figure 6: FLOPs & e2e speedup vs user-specified acceleration budget.

Fig 6 demonstrates that CUBIC strategy effectively reduces flops in the selective phase, while the sparse executor efficiently transforms this into an end-to-end linear speedup. The speedup is lower than the user target because the target only applies to the selective phase; the conversion from the selective-phase target to the overall compute fraction is given in Sec. 4.2.

5.5 Case Study: Components visualization

Due to space constraints, detailed visualizations of the dynamic pipeline are deferred to Appendix C. At a target speedup of $3\times$, CUBIC executes five anchor steps. The scheduling exhibits a clear temporal evolution: early steps show high and heterogeneous curvature, leading to diverse interval allocations; middle steps capture localized high-curvature windows; and late steps converge to uniform curvature with consistent caching strategies. Throughout the process, the sparse ratio is stably maintained around 0.33, and the step latency closely tracks the active ratio.

6 Conclusion & Limitation

In this paper, we introduced CUBIC, a training-free dynamic caching framework for efficient Flow Matching inference. By grounding the caching mechanism in the geometric properties of the velocity field, CUBIC adaptively partitions latents and allocates update intervals that minimize a curvature-based error surrogate under a fixed compute budget. Comprehensive evaluations across diverse generation tasks demonstrate that CUBIC establishes a superior quality-efficiency Pareto frontier compared to existing heuristic baselines. Nevertheless, our theoretical analysis is based on a forward-Euler abstraction of the sampling trajectory. While this offers a tractable link between curvature and cache-induced error, practical diffusion and flow-matching samplers often use multi-step, higher-order, or predictor–corrector updates, whose error constants and propagation patterns may differ. Therefore, our curvature-based scheduling rule should be viewed as a principled local approximation rather than a solver-agnostic guarantee, and extending the analysis to more general samplers is left for future work. Our bound also assumes deterministic ODE sampling, the standard inference path of all evaluated models. SDE-formulated models can use the corresponding probability-flow ODE and apply CUBIC directly, but native SDE sampling requires stochastic error bounds that account for cached drift interacting with diffusion, which we leave to future work. CUBIC further fixes ρ across the trajectory for predictable compute; a time-varying budget is promising since curvature evolves over steps, but requires predicting future curvature without extra overhead. The anchor horizon and block granularity are empirical, model-dependent choices rather than derived optima.

Acknowledgements and Disclosures

We would like to thank the anonymous reviewers for their insightful comments. This work is supported by the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (JYB2025XDXM910), NSFC for Distinguished Young Scholar under Grant 62225206, National Natural Science Foundation of China under Grants 62532006, 62495062, U23A600, and Beijing Natural Science Foundation under Grants L242017, L243001, L2603007. Jidong Zhai is the corresponding author of this paper. The authors declare no competing interests.

References

[1] Muhammad Adnan, Nithesh Kurella, Akhil Arunkumar, and Prashant J. Nair. Foresight: Adaptive layer reuse for accelerated and high-quality text-to-video generation, 2025.

[2] Manuel Brack, Sudeep Katakol, Felix Friedrich, Patrick Schramowski, Hareesh Ravi, Kristian Kersting, and Ajinkya Kale. How to train your text-to-image model: Evaluating design choices for synthetic training captions. *arXiv preprint arXiv:2506.16679*, 2025.

[3] Siyu Cao, Hangting Chen, Peng Chen, Yiji Cheng, Yutao Cui, Xinchi Deng, Ying Dong, Kipper Gong, Tianpeng Gu, Xiusen Gu, et al. Hunyuanimage 3.0 technical report. *arXiv preprint arXiv:2509.23951*, 2025.

[4] Pengtao Chen, Mingzhu Shen, Peng Ye, Jianjian Cao, Chongjun Tu, Christos-Savvas Bouganis, Yiren Zhao, and Tao Chen. Delta-dit: A training-free acceleration method tailored for diffusion transformers, 2024.

[5] Yuyang Chen, Linqian Zeng, Yijin ZHou, Hengjie Li, and Jidong Zhai. Jano: Adaptive diffusion generation with early-stage convergence awareness, 2026.

[6] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis, 2024.

[7] Huanlin Gao, Ping Chen, Fuyuan Shi, Ruijia Wu, Yantao Li, 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. 2026.

[8] Zhengyang Geng, Mingyang Deng, Xingjian Bai, J. Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling, 2025.

[9] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[10] Mingzhen Huang, Jialing Cai, Shan Jia, Vishnu Suresh Lokhande, and Siwei Lyu. Paralleledits: Efficient multi-aspect text-driven image editing with attention grouping. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 22569–22595. Curran Associates, Inc., 2024.

[11] Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, Yaohui Wang, Xinyuan Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. VBench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[12] Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, Kathrina Wu, Qin Lin, Junkun Yuan, Yanxin Long, Aladdin Wang, Andong Wang, Changlin Li, Duojun Huang, Fang Yang, Hao Tan, Hongmei Wang, Jacob Song, Jiawang Bai, Jianbing Wu, Jinbao Xue, Joey Wang, Kai Wang, Mengyang Liu, Pengyu Li, Shuai Li, Weiyan Wang, Wenqing Yu, Xinchi Deng, Yang Li, Yi Chen, Yutao Cui, Yuanbo Peng, Zhentao Yu, Zhiyu He, Zhiyong Xu, Zixiang Zhou, Zunnan Xu, Yangyu Tao, Qinglin Lu, Songtao Liu, Daquan Zhou, Hongfa Wang, Yong Yang, Di Wang, Yuhong Liu, Jie Jiang, and Caesar Zhong. Hunyuanvideo: A systematic framework for large video generative models. *CoRR*, abs/2412.03603, 2024.

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

[14] Shanchuan Lin, Anran Wang, and Xiao Yang. Sdxl-lightning: Progressive adversarial diffusion distillation, 2024.

[15] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling, 2023.

[16] 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. *CoRR*, abs/2411.19108, 2024.

[17] Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Junjie Chen, and Linfeng Zhang. From reusing to forecasting: Accelerating diffusion models with taylorseers. *arXiv preprint arXiv:2503.06923*, 2025.

[18] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow, 2022.

[19] Ziming Liu, Yifan Yang, Chengruidong Zhang, Yiqi Zhang, Lili Qiu, Yang You, and Yuqing Yang. Region-adaptive sampling for diffusion transformers, 2025.

[20] Yuhang Ma, Xiaoshi Wu, Keqiang Sun, and Hongsheng Li. Hpsv3: Towards wide-spectrum human preference score. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 15086–15095, 2025.

[21] Zehong Ma, Longhui Wei, Feng Wang, Shiliang Zhang, and Qi Tian. Magcache: Fast video generation with magnitude-aware cache, 2025.

[22] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik P. Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On Distillation of Guided Diffusion Models. *arXiv e-prints*, page arXiv:2210.03142, October 2022.

[23] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*, pages 4172–4182. IEEE, 2023.

[24] Xiangyu Peng, Zangwei Zheng, Chenhui Shen, Tom Young, Xinying Guo, Binluo Wang, Hang Xu, Hongxin Liu, Mingyan Jiang, Wenjun Li, Yuhui Wang, Anbang Ye, Gang Ren, Qianran Ma, Wanying Liang, Xiang Lian, Xiwen Wu, Yuting Zhong, Zhuangyan Li, Chaoyu Gong, Guojun Lei, Leijun Cheng, Limin Zhang, Minghao Li, Ruijie Zhang, Silan Hu, Shijie Huang, Xiaokang Wang, Yuanheng Zhao, Yuqi Wang, Ziang Wei, and Yang You. Open-sora 2.0: Training a commercial-level video generation model in $200k. *arXiv preprint arXiv:2503.09642*, 2025.

[25] Axel Sauer, Dominik Lorenz, Andreas Blattmann, and Robin Rombach. Adversarial diffusion distillation, 2023.

[26] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models, 2022.

[27] Super Intelligence Team. Firered-image-edit-1.0 technical report. 2026.

[28] 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.

[29] Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, Yuxiang Chen, Zecheng Tang, Zekai Zhang, Zhengyi Wang, An Yang, Bowen Yu, Chen Cheng, Dayiheng Liu, Deqing Li, Hang Zhang, Hao Meng, Hu Wei, Jingyuan Ni, Kai Chen, Kuan Cao, Liang Peng, Lin Qu, Minggang Wu, Peng Wang, Shuting Yu, Tingkun Wen, Wensen Feng, Xiaoxiao Xu, Yi Wang, Yichang Zhang, Yongqiang Zhu, Yujia Wu, Yuxuan Cai, and Zenan Liu. Qwen-image technical report, 2025.

[30] Haocheng Xi, Shuo Yang, Yilong Zhao, Chenfeng Xu, Muyang Li, Xiuyu Li, Yujun Lin, Han Cai, Jintao Zhang, Dacheng Li, Jianfei Chen, Ion Stoica, Kurt Keutzer, and Song Han. Sparse videogen: Accelerating video diffusion transformers with spatial-temporal sparsity. *CoRR*, abs/2502.01776, 2025.

[31] Shuo Yang, Haocheng Xi, Yilong Zhao, Muyang Li, Jintao Zhang, Han Cai, Yujun Lin, Xiuyu Li, Chenfeng Xu, Kelly Peng, et al. Sparse videogen2: Accelerate video generation with sparse attention via semantic-aware permutation. *arXiv preprint arXiv:2505.18875*, 2025.

[32] Zhihang Yuan, Hanling Zhang, Lu Pu, Xuefei Ning, Linfeng Zhang, Tianchen Zhao, Shengen Yan, Guohao Dai, and Yu Wang. DiTFastattn: Attention compression for diffusion transformer models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[33] Weixuan Zeng, Pengcheng Wei, Huaiqing Wang, Boheng Zhang, Jia Sun, Dewen Fan, Lin HE, Long Chen, Qianqian Gan, Fan Yang, and Tingting Gao. Omnidit: Extending diffusion transformer to omni-vton framework, 2026.

[34] Hui Zhang, Tingwei Gao, Jie Shao, and Zuxuan Wu. Blockdance: Reuse structurally similar spatio-temporal features to accelerate diffusion transformers, 2025.

[35] Jintao Zhang, Haofeng Huang, Pengle Zhang, Jia Wei, Jun Zhu, and Jianfei Chen. Sageattention2: Efficient attention with thorough outlier smoothing and per-thread int4 quantization. In *International Conference on Machine Learning (ICML)*, 2025.

[36] Jintao Zhang, Jia Wei, Pengle Zhang, Xiaoming Xu, Haofeng Huang, Haoxu Wang, Kai Jiang, Jun Zhu, and Jianfei Chen. Sageattention3: Microscaling fp4 attention for inference and an exploration of 8-bit training. *arXiv preprint arXiv:2505.11594*, 2025.

[37] Jintao Zhang, Jia Wei, Pengle Zhang, Jun Zhu, and Jianfei Chen. Sageattention: Accurate 8-bit attention for plug-and-play inference acceleration. In *International Conference on Learning Representations (ICLR)*, 2025.

[38] Jintao Zhang, Xiaoming Xu, Jia Wei, Haofeng Huang, Pengle Zhang, Chendong Xiang, Jun Zhu, and Jianfei Chen. Sageattention2++: A more efficient implementation of sageattention2. *arXiv preprint arXiv:2505.21136*, 2025.

[39] Xuanlei Zhao, Xiaolong Jin, Kai Wang, and Yang You. Real-time video generation with pyramid attention broadcast, 2024.

[40] Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching, 2025.

A Theoretical Justification of CUBIC

This section provides the theoretical justification for the curvature-aware caching policy used in CUBIC. We first analyze the state error induced by cache reuse and show that it is jointly controlled by local trajectory curvature and the cache reuse interval. We then convert this error model into a block-wise interval allocation problem under a global computation budget and derive the optimal reuse interval. Finally, we show how the required curvature can be approximated online using endpoint prediction drift without additional model forward passes.

A.1 Curvature Approximation

The optimal interval derived in Sec. 4.2 depends on the token-wise curvature $\kappa^{(i)}$. Directly computing $\kappa^{(i)}(t) = \|dv_\theta^{(i)}/dt\|_2$, however, requires Jacobian or higher-order derivative information and is therefore expensive for large-scale Flow Matching models. CUBIC instead uses endpoint prediction drift as an online curvature proxy.

For a spatial token indexed by i, define the one-step endpoint prediction at time t as

$$\hat{x}_1^{(i)}(t) = x^{(i)}(t) + (1-t)v_\theta^{(i)}(x(t), t). \tag{14}$$

This quantity is the endpoint that would be reached at $t = 1$ if the current velocity were kept constant. If the local trajectory is nearly linear, endpoint predictions at nearby times should remain almost unchanged. In contrast, a curved trajectory produces a noticeable endpoint drift.

Taking the total derivative of Eq. (14) along the exact trajectory gives

$$\begin{aligned}\frac{d\hat{x}_1^{(i)}}{dt} &= \frac{dx^{(i)}}{dt} - v_\theta^{(i)}(x(t), t) + (1-t)\frac{dv_\theta^{(i)}}{dt} \\ &= (1-t)\frac{dv_\theta^{(i)}}{dt}.\end{aligned} \tag{15}$$

Let $t_{a_{m-1}}$ and t_{a_m} be two consecutive anchor steps, and denote $\Delta t_{a_m} = t_{a_m} - t_{a_{m-1}}$. A first-order Taylor expansion gives

$$\hat{\delta}^{(i,m)} = \hat{x}_1^{(i)}(t_{a_m}) - \hat{x}_1^{(i)}(t_{a_{m-1}}) \approx (1 - t_{a_m})\frac{dv_\theta^{(i)}}{dt}(t_{a_m})\Delta t_{a_m}. \tag{16}$$

Taking norms yields

$$\left\|\hat{\delta}^{(i,m)}\right\|_2 \approx (1 - t_{a_m})\Delta t_{a_m}\kappa^{(i)}(t_{a_m}). \tag{17}$$

Therefore, the normalized endpoint drift

$$\hat{\kappa}^{(i)} = \frac{\left\|\hat{\delta}^{(i,m)}\right\|_2}{(1 - t_{a_m})\Delta t_{a_m}} \tag{18}$$

serves as an online approximation of $\kappa^{(i)}(t_{a_m})$.

This proxy uses only quantities already available at anchor steps, namely the current latent state and the model-predicted velocity. It therefore requires no additional model forward passes.

A.2 Error Modeling of Cache Reuse

We first derive the local state error induced by cache reuse at the position level. Consider a block b that is fully recomputed at time t_0, producing the cached velocity

$$v_0^{(b)} = v_\theta^{(b)}(x(t_0), t_0). \tag{19}$$

If this velocity is reused for the next τ_b solver steps, the local approximate dynamics follow a zero-order hold:

$$\frac{dx_{\text{cache}}^{(b)}(t)}{dt} = v_0^{(b)}. \tag{20}$$

Let $x_{\text{exact}}^{(b)}(t)$ denote the exact trajectory. The cache-induced state error is

$$\varepsilon_b(t) = x_{\text{cache}}^{(b)}(t) - x_{\text{exact}}^{(b)}(t) = \int_{t_0}^{t} \left[v_0^{(b)} - v_\theta^{(b)}(x(t'), t') \right] dt'. \tag{21}$$

We quantify the local trajectory curvature at block b by the rate of change of the velocity field along the exact trajectory:

$$\kappa_b(t) = \left\| \frac{dv_\theta^{(b)}}{dt}(x(t), t) \right\|_2. \tag{22}$$

Assume that $\kappa_b(t) \le \bar{\kappa}_b$ within the reuse window. Then the velocity mismatch satisfies

$$\left\| v_0^{(b)} - v_\theta^{(b)}(x(t), t) \right\|_2 \le \bar{\kappa}_b (t - t_0). \tag{23}$$

Integrating over a reuse window of length $\tau_b \Delta t$ gives

$$\|\varepsilon_b(t_0 + \tau_b \Delta t)\|_2 \le \frac{1}{2} \bar{\kappa}_b \tau_b^2 \Delta t^2. \tag{24}$$

The measured cache-induced error follows an approximate power-law dependence on both the reuse interval τ and the local curvature proxy κ, supporting the error bound in Eq. (24).

Therefore, cache reuse introduces a state error that grows linearly with local curvature and quadratically with the reuse interval. Taking the squared norm as the cache-loss surrogate gives

$$\ell_b(\tau_b) = \|\varepsilon_b(t_0 + \tau_b \Delta t)\|_2^2 \lesssim \frac{1}{4} \bar{\kappa}_b^2 \tau_b^4 \Delta t^4. \tag{25}$$

Dropping shared constants yields the default surrogate

$$\ell_b(\tau_b) \propto \kappa_b^2 \tau_b^4. \tag{26}$$

More generally, we write

$$\ell_b(\tau_b) = a_b \kappa_b^\alpha \tau_b^\beta, \tag{27}$$

where the theoretical default is $(\alpha, \beta) = (2, 4)$.

A.3 Optimal Cache Reuse Interval under a Target Speedup

We next derive the optimal block-wise reuse interval under a target speedup. Let τ_b denote the update interval of block b, and let $\mathcal{B}_b$ be the set of tokens contained in this block. Since different blocks may contain different numbers of tokens, both the cache-induced error and the computation reduction should be weighted by the block size $|\mathcal{B}_b|$.

Using the generalized cache-loss surrogate from Eq. (27), we consider the following block-weighted optimization problem:

$$\min_{\{\tau_b\}_{b=1}^B} \sum_{b=1}^{B} |\mathcal{B}_b| \kappa_b^\alpha \tau_b^\beta, \qquad \text{s.t.} \quad \sum_{b=1}^{B} \frac{|\mathcal{B}_b|}{N} \frac{1}{\tau_b} = \frac{1}{\rho}, \quad \tau_b \ge 1, \tag{28}$$

where $N = \sum_{b=1}^{B} |\mathcal{B}_b|$ is the total number of spatial tokens, ρ is the target speedup, and κ_b is the block-wise curvature estimate. The constraint states that the block-size-weighted average update rate equals the reciprocal of the target speedup.

Ignoring integer and box constraints, the Lagrangian is

$$\mathcal{L} = \sum_{b=1}^{B} |\mathcal{B}_b| \kappa_b^\alpha \tau_b^\beta + \lambda \left(\sum_{b=1}^{B} \frac{|\mathcal{B}_b|}{N} \frac{1}{\tau_b} - \frac{1}{\rho} \right). \tag{29}$$

Setting $\partial \mathcal{L} / \partial \tau_b = 0$ gives

$$|\mathcal{B}_b| \beta \kappa_b^\alpha \tau_b^{\beta - 1} - \lambda \frac{|\mathcal{B}_b|}{N} \tau_b^{-2} = 0. \tag{30}$$

The block size $|\mathcal{B}_b|$ appears in both the objective and the constraint, and therefore cancels out in the first-order optimality condition. We obtain

$$\tau_b^\star \propto \kappa_b^{-\alpha/(\beta+1)}. \tag{31}$$

With the squared-error default $(\alpha, \beta) = (2, 4)$, this becomes

$$\tau_b^\star \propto \kappa_b^{-2/5}. \tag{32}$$

Thus, high-curvature blocks are refreshed more frequently, whereas low-curvature blocks can safely reuse cached states for longer intervals. Because the allocation depends on (α, β) only through $\gamma = \alpha/(\beta + 1)$, matching the raw exponents is not required: the exponents fitted from cache-error probes, $(1.81, 3.426)$, give $\gamma = 0.409$, and yield nearly the same schedule as $\gamma = 2/5$ (Appendix E.2).

To satisfy the target speedup constraint, we normalize the proportional rule. Let $\gamma = \alpha/(\beta + 1)$. Then the continuous interval is

$$\tilde{\tau}_b = \rho \cdot \frac{\frac{1}{N}\sum_{j=1}^{B} |\mathcal{B}_j| \kappa_j^\gamma}{\kappa_b^\gamma}. \tag{33}$$

In practice, we use the squared-error default $\gamma = 2/5$ and discretize the interval as

$$\tau_b = \text{round}\left(\rho' \cdot \frac{\frac{1}{N}\sum_{j=1}^{B} |\mathcal{B}_j| \kappa_j^{2/5}}{\kappa_b^{2/5} + \varepsilon}\right), \tag{34}$$

where ρ' absorbs the adjustment caused by rounding and clipping, and ε prevents numerical instability for near-zero curvature. We set $\tau_{\min} = 1$ so that high-curvature blocks can be recomputed at every step.

B Details of Partitioning Strategy

In Sec. 4.2, we introduced the morphology-guided adaptive partitioning strategy. The preferred partitioning scheme is model-dependent (Appendix E.4). For long-sequence, high-overhead models such as Wan (sequence length 32760), we adopt coarse morphology-guided partitioning, which improves fidelity over a uniform grid with a matched number of blocks; for shorter, fast-inference models such as Flux (sequence length 4096), a simple fixed-size partitioning achieves comparable quality with much lower overhead. We provide the detailed algorithmic steps of the morphology-guided strategy here. The input is the raw token-space curvature map $\kappa \in \mathbb{R}^{F_t \times H_t \times W_t}$ computed at the current anchor step.

1. Spatial-Temporal Reduction Since the spatial partition should remain stable across the subsequent selective steps within the same anchor interval, we first aggregate the curvature information along the temporal dimension to obtain a 2D spatial curvature map:

$$\bar{\kappa}_{h,w} = \frac{1}{F_t}\sum_{f=1}^{F_t} \kappa_{f,h,w}. \tag{35}$$

2. Geometric Prior Extraction We binarize the spatial map to distinguish complex (foreground) regions from flat (background) regions. Given a high percentile threshold q_f (e.g., $q_f = 0.85$), we construct the initial foreground mask:

$$M_{h,w} = \mathbb{1}\left[\bar{\kappa}_{h,w} \geq \text{Quantile}_{q_f}(\bar{\kappa})\right]. \tag{36}$$

Raw thresholding often yields spatial noise or fragmented boundaries. To regularize the partition boundary without any model forward passes, we apply a sequence of standard morphological operations to M: opening and closing (to remove salt-and-pepper noise and fill small holes), followed by dilation and erosion (to smooth jagged edges), and finally small connected-component filtering (to discard degenerate micro-regions). This yields a cleaned, structurally stable mask $\tilde{M}$.

3. Recursive Spatial Bisection Given the geometric prior $\tilde{M}$, we partition the entire 2D token grid Ω into non-overlapping rectangles $\mathcal{P} = \{\mathcal{B}_b\}$. We define two target block sizes: a fine granularity S_{fg} for foreground and a coarse granularity S_{bg} for background ($S_{\text{fg}} < S_{\text{bg}}$).

The algorithm starts from the root region $R = \Omega$ and recursively evaluates it. For any current rectangular region R, we calculate its foreground ratio $p_f = \frac{\sum \tilde{M}_{h,w \in R}}{|R|}$ and its intra-region curvature Coefficient of Variation (CV). The region R is **split** if any of the following conditions are met:

1. Mixed Composition: p_f falls within a mixed interval $[p_{\text{low}}, p_{\text{high}}]$ (e.g., $[0.2, 0.8]$), indicating it contains a blend of complexities.
2. High Internal Variance: The curvature CV within R exceeds a consistency threshold τ_{cv}.
3. Size Exceeds Target: The height or width of R exceeds the target size determined by its dominant property (i.e., S_{fg} if $p_f > 0.5$, else S_{bg}).

If split, we choose the splitting axis (horizontal or vertical) that minimizes the foreground variance across the split, and bisect R. If none of the criteria are met, R is finalized as a block $\mathcal{B}_b \in \mathcal{P}$. The output satisfies:

$$\bigcup_{b=1}^{\mathcal{B}} \mathcal{B}_b = \Omega, \quad \mathcal{B}_b \cap \mathcal{B}_{b'} = \varnothing \ (b \neq b'). \tag{37}$$

4. Aggregation and Fallback Once $\mathcal{P}$ is fixed, we aggregate the token-wise curvature into a block-wise scheduling signal using the spatial mean:

$$\kappa_b = \frac{1}{|\mathcal{B}_b|} \sum_{i \in \mathcal{B}_b} \kappa_i. \tag{38}$$

This mean aggregation is stable precisely because Step 3 guarantees that any block containing high-curvature outliers would have been further subdivided. To ensure robustness in edge cases (e.g., entirely uniform curvature maps), a standard uniform grid partitioning using S_{bg} remains available as an explicit fallback configuration.

C CUBIC Case Study

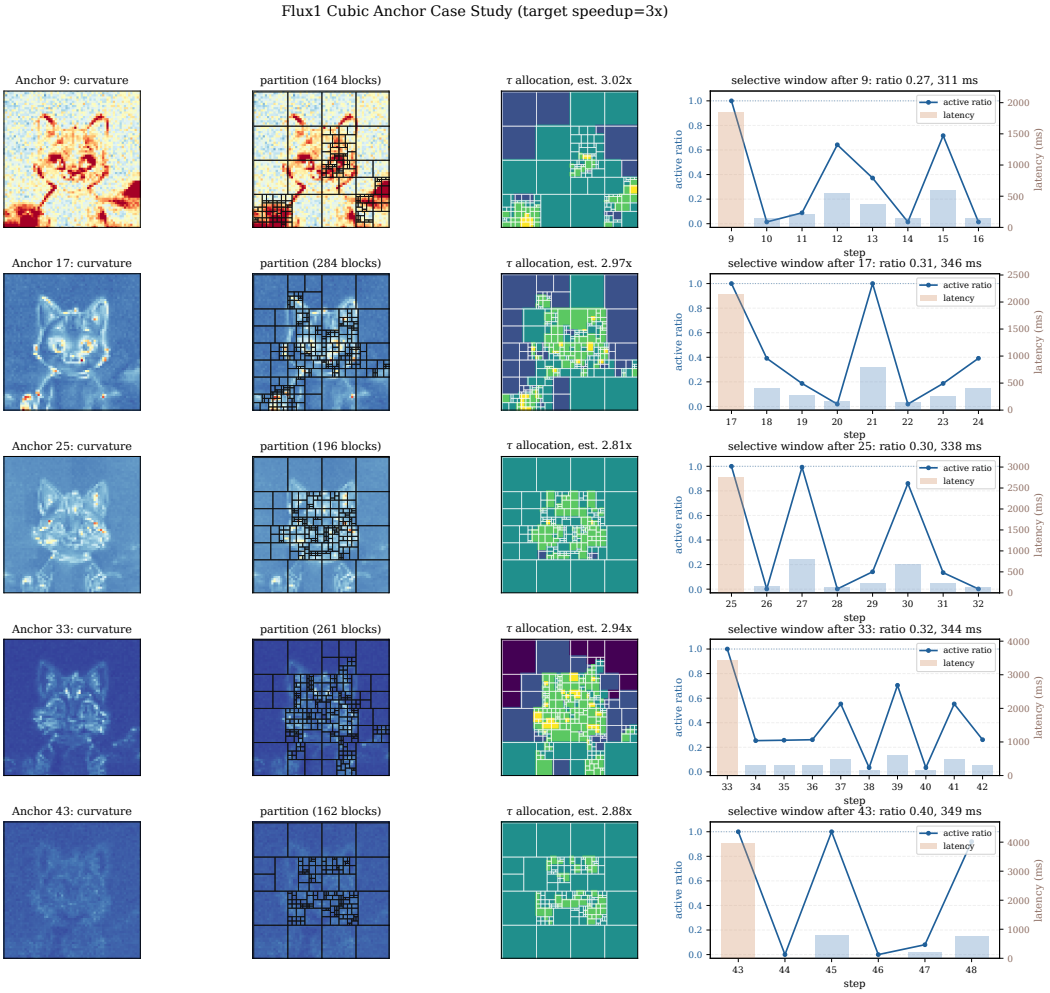

Figure 7: CUBIC case study on Flux.1 with target speedup 3×.

D Detailed Results of CUBIC Evaluation

We report the detailed parameter settings and performance of CUBIC in Table 3. For completeness, Table 4 provides the corresponding configurations and results of other baseline methods.

Evaluation protocol. We randomly sample 500 prompts from t2i-diversity-eval and 200 examples from PIE-Bench++, and use 30 prompts from each VBench category. Latency is measured end to end under each task's default inference settings. The reference for PSNR, SSIM, and LPIPS is the output of the uncached full model under the same prompt, random seed, and initial noise, rather than dataset ground truth; for image editing, the reference also uses the same source image and editing instruction. The standard and fast settings of CUBIC use $\rho = 4$ and $\rho = 8$ on Flux and FireRed, and $\rho = 2$ and $\rho = 6$ on Wan2.1.

FireRed baselines. FireRed has no official implementation of the caching baselines. We adapt TeaCache, TaylorSeer, PAB, and ToCa from their official open-source implementations on analogous models, preserving their respective similarity-probe, Taylor-prediction, attention-caching, and token-selection mechanisms. For fairness, all methods use identical FireRed weights, scheduler, precision, inputs, prompts, seeds, CFG scale, and denoising steps. The cache state is reset before every sample, and all outputs are evaluated against the same uncached baseline with a single timing and quality pipeline. The complete FireRed adaptations, configurations, and run scripts are included in our code release.

Table 3: Parameters setting and performance of CUBIC

Model	$\boldsymbol{\rho}$	Speedup ↑	PSNR ↑	SSIM ↑	LPIPS ↓	FID/FVD ↓
Flux-1	2	1.38×	32.03	0.9445	0.0409	12.51
	4	1.89×	27.78	0.8998	0.0946	27.81
	6	2.22×	26.04	0.8691	0.1299	37.14
	8	2.40×	25.14	0.8457	0.1730	47.32
	10	2.51×	24.45	0.8282	0.1964	52.82
FireRed	2	1.76×	33.85	0.9711	0.0268	10.91
	4	2.14×	30.67	0.9538	0.0468	18.55
	6	2.23×	28.74	0.9310	0.0769	27.15
	8	2.33×	27.81	0.9201	0.0907	32.18
	10	2.50×	27.06	0.9080	0.1059	38.35
Wan-1.3B	2	1.63×	32.76	0.9481	0.0317	7.912
	4	2.30×	27.99	0.8990	0.0728	21.14
	6	2.72×	26.39	0.8740	0.0986	31.76
	8	3.04×	25.50	0.8501	0.1236	45.08
	10	3.23×	24.65	0.8284	0.1437	62.48

Table 4: Parameters setting and performance of baselines.

Model	Method	Speedup ↑	PSNR ↑	SSIM ↑	LPIPS ↓	FID/FVD ↓
Flux-1	ToCa(4)	1.69×	20.34	0.7647	0.2414	61.97
	ToCa(5)	1.81×	19.68	0.7493	0.2726	70.76
	ToCa(6)	1.93×	18.88	0.7094	0.3164	78.10
	TeaCache(0.25)	1.86×	22.88	0.8514	0.1490	45.87
	TeaCache(0.4)	2.52×	19.66	0.7720	0.2481	66.04
	TeaCache(0.6)	3.31×	17.66	0.7279	0.3155	89.86
	TaylorSeer(N2)	1.70×	26.29	0.8920	0.0994	26.80
	TaylorSeer(N3)	2.25×	21.42	0.8180	0.1796	47.53
	TaylorSeer(N4)	2.66×	19.27	0.7606	0.2497	63.59
	PAB(4)	1.62×	22.88	0.8373	0.1613	47.83
	PAB(8)	1.81×	20.37	0.7702	0.2574	77.37
	PAB(12)	1.90×	19.47	0.7344	0.3293	106.65
FireRed	ToCa(3)	1.50×	24.17	0.8736	0.1103	49.84
	ToCa(4)	1.76×	22.57	0.8367	0.1471	62.58
	ToCa(5)	1.95×	21.09	0.7884	0.1779	67.25
	TeaCache(0.1)	1.51×	27.49	0.9363	0.0562	25.55
	TeaCache(0.15)	2.04×	25.65	0.9183	0.0758	31.00
	TeaCache(0.2)	2.31×	23.80	0.8955	0.0995	41.03
	TaylorSeer(N2)	1.86×	25.12	0.9102	0.0837	33.70
	TaylorSeer(N3)	2.64×	21.17	0.8356	0.1498	50.38
	TaylorSeer(N4)	3.35×	18.96	0.7774	0.1987	64.84
	PAB(6)	1.61×	25.82	0.9112	0.0731	33.92
	PAB(8)	1.65×	24.39	0.8888	0.0929	39.41
	PAB(12)	1.68×	22.45	0.8530	0.1312	54.07
	PAB(16)	1.72×	20.87	0.9198	0.1799	67.82
Wan1.3B	ToCa(242)	1.69×	19.60	0.7486	0.2273	78.95
	ToCa(333)	2.21×	20.66	0.7684	0.1949	66.02
	ToCa(363)	2.31×	19.19	0.7383	0.2405	84.04
	TeaCache(0.05)	1.65×	30.27	0.9270	0.0454	11.72
	TeaCache(0.07)	1.75×	26.38	0.8684	0.0933	24.85
	TeaCache(0.08)	2.15×	24.51	0.8405	0.1206	35.26
	TaylorSeer(N2)	1.91×	18.29	0.7225	0.2504	94.59
	TaylorSeer(N3)	2.59×	15.80	0.6410	0.3521	137.06
	TaylorSeer(N4)	3.19×	14.04	0.5837	0.4406	181.41
	PAB(2,5)	1.67×	24.06	0.8370	0.1212	36.82
	PAB(4,5)	2.17×	22.83	0.8127	0.1530	52.37
	PAB(6,6)	2.50×	22.36	0.8000	0.1670	57.78

E Additional Experiments

This section reports a direct comparison with Jano and ablations on the system choices of CUBIC. Unlike the main experiments, these runs use an NVIDIA H20 GPU. Speedup is normalized by the uncached model on the same code path as each method, and PSNR, SSIM, and LPIPS are computed against the uncached output with the same prompt and seed. Entries with $\pm$ report the mean and standard deviation over samples. Here ρ follows the convention of Sec. 4.2, i.e., the target speedup of the selective phase.

E.1 Direct Comparison with Jano

Protocol. To avoid selecting the operating point of CUBIC on the evaluation data, we sample two disjoint prompt sets with a fixed random seed. For Flux, we draw 4 tuning prompts and 16 test prompts from t2i-diversity-eval [2]; for Wan2.1-1.3B, we draw 3 tuning prompts and 12 test prompts from the VBench prompt list [11]. Jano uses its official default configuration throughout. On the tuning prompts with seed 42, we sweep the target speedup ρ of CUBIC (and, on Wan, the number of full-compute warm-up steps), and keep the configurations that are faster than Jano and better on all three fidelity metrics. These configurations are then evaluated once on the test prompts with seeds 42 and 43, giving 32 Flux and 24 Wan samples. Flux uses 1024×1024 resolution, 50 steps, and guidance scale 3.5, and both methods use 7 full-compute steps at the beginning and 2 at the end. Wan generates 81 frames at 832×480 with 50 steps; Jano uses its default 6 and 4 full-compute steps, and CUBIC uses 11 and 1. Each Flux run performs one untimed warm-up generation before timing. Jano on Flux records only the aggregate latency over all samples, so its speedup has no standard deviation.

Table 5 shows that on Flux both configurations of CUBIC are faster than Jano, by 12% and 20%, and better on all three fidelity metrics. On Wan, CUBIC with $\rho = 6$ is better on all three fidelity metrics at a comparable speed: its mean speedup is 4% higher, but it is faster than Jano on only 16 of 24 samples, so we do not claim a speed advantage. Under a paired Wilcoxon signed-rank test, every

Table 5: Direct comparison with Jano on held-out test prompts (Flux: 16 prompts $\times$ 2 seeds; Wan2.1-1.3B: 12 prompts $\times$ 2 seeds). Wins count the samples on which CUBIC beats Jano under the same prompt and seed, listed for PSNR/SSIM/LPIPS.

Model	Method	Speedup ↑	PSNR ↑	SSIM ↑	LPIPS ↓	Wins
Flux	Jano	2.18×	21.93±4.03	0.788±0.103	0.225±0.096	–
	CUBIC ($\rho = 5$)	2.43±0.16×	**23.47±4.12**	**0.842±0.086**	**0.145±0.075**	29/30/32
	CUBIC ($\rho = 6$)	**2.63±0.12**×	22.79±4.05	0.823±0.097	0.171±0.084	27/29/32
Wan2.1-1.3B	Jano	2.28±0.21×	20.64±3.03	0.766±0.101	0.147±0.046	–
	CUBIC ($\rho = 6$)	**2.36±0.05**×	23.78±3.04	0.843±0.062	0.096±0.033	24/22/23
	CUBIC ($\rho = 5$)	2.22±0.03×	**24.47±3.21**	**0.860±0.056**	**0.084±0.030**	24/23/24

fidelity improvement in Table 5 has $p < 10^{-4}$, and the speedup difference on Wan has $p = 0.10$. CUBIC also has a much smaller speedup variance on Wan, since its compute is fixed by ρ, whereas Jano's depends on the content.

Fig. 8 and Fig. 9 show the seed-42 sample of the first three test prompts, so the examples are not selected by quality; in the first Flux example, Jano has a slightly higher PSNR. The two methods share the same uncached reference, shown in the first row.



Figure 8: Flux outputs of the uncached model, Jano, and CUBIC ($\rho = 5$) under identical test prompts and seed 42. Numbers are per-image PSNR against the uncached output.

E.2 Sensitivity to the Scheduling Exponent

Under the budget constraint, the interval allocation depends on the error exponents only through $\gamma = \alpha/(\beta + 1)$ (Eq. (31)). We fix the prompt, the 4×4 partition, the full-compute boundary, and

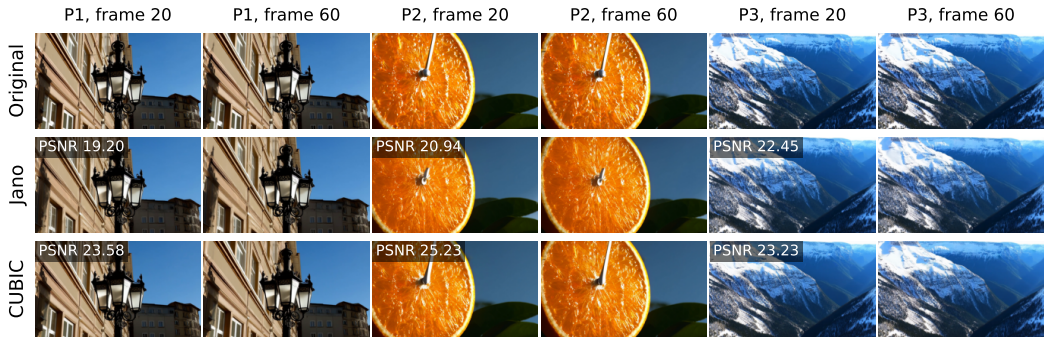

Figure 9: Wan2.1-1.3B outputs of the uncached model, Jano, and CUBIC ($\rho = 6$) under identical test prompts and seed 42, showing frames 20 and 60 of each video. Numbers are per-video PSNR averaged over all frames against the uncached output.

$\rho = 4$, and vary only γ over seeds 42, 43, and 44 on Flux. The theoretical $\gamma = 0.400$ and the fitted $\gamma = 0.409$ assign identical intervals at the first anchor step, their interval histograms over all anchors differ by a single count, and their quality metrics coincide within display precision. Over $\gamma \in [0.3, 0.6]$, speedup changes by about 1.4% and LPIPS by 0.0066. This supports the equivalence of the theoretical and fitted exponents and low sensitivity around them, but not a unique optimum.

Table 6: Budget-controlled sweep of the scheduling exponent γ on Flux ($\rho = 4$, 3 seeds).

γ	Source	Active ratio	Latency (s) ↓	Speedup ↑	PSNR ↑	SSIM ↑	LPIPS ↓
0.300	control	0.4500	18.61	2.021×	32.87	0.9737	0.0302
0.400	theoretical $(2, 4)$	0.4458	18.67	2.014×	32.14	0.9702	0.0344
0.409	fitted $(1.81, 3.426)$	0.4458	18.63	2.019×	32.13	0.9702	0.0344
0.500	control	0.4417	18.60	2.022×	31.88	0.9692	0.0350
0.600	control	0.4304	18.35	2.049×	31.75	0.9680	0.0368

E.3 Anchor Horizon

We compare the default selective-phase length $2\tau_{\max}$ with $1\tau_{\max}$, $3\tau_{\max}$, and a fixed gap of 7 steps, which is close to the mean gap of the default rule. All policies use $\rho = 3$ and a 3×3 grid, and are evaluated on the same eight prompt–seed samples on Flux. The rank correlation measures how well the block curvature ordering at one anchor step agrees with that at the next one.

Table 7: Anchor horizon ablation on Flux ($\rho = 3$, 8 samples).

Anchor policy	Mean gap	Active ratio	Speedup ↑	PSNR ↑	Curvature rank corr.
$1\tau_{\max}$	3.35	0.646±0.045	1.520±0.114×	34.43	0.750±0.027
$2\tau_{\max}$	6.69	0.603±0.014	1.630±0.047×	34.14	0.702±0.021
$3\tau_{\max}$	9.63	0.582±0.004	1.690±0.021×	34.09	0.650±0.035
Fixed gap 7	7.00	0.640±0.003	1.538±0.015×	34.88	0.704±0.034

Longer dynamic horizons increase speedup while the curvature ordering becomes less consistent between consecutive anchors, yet PSNR stays stable. This indicates that re-estimating curvature at each anchor corrects stale schedules before their errors persist. A fixed gap cannot follow the allocated intervals: among the 34 windows of the $2\tau_{\max}$ rule, a gap of 7 would re-plan too early for 11 and too late for 23. It therefore realizes a higher active ratio than the target and a lower speedup. Its higher PSNR together with lower speed reflects a different point on the latency–quality trade-off rather than dominance of either policy. We use $2\tau_{\max}$ as a balance between robustness and acceleration.

E.4 Block Granularity

Table 8 varies the partition at $\rho = 3$. On Flux, we use the same eight samples as Appendix E.3. On Wan2.1-1.3B, we use a single prompt and seed with 17 frames and 20 steps, and compare adaptive partitions against uniform grids with nearly the same number of blocks, so that granularity and adaptivity are not confounded. Partition overhead is the time spent constructing the partition per anchor step.

Table 8: Block granularity ablation at $\rho = 3$.

Model	Partition	Blocks	Speedup ↑	PSNR ↑	Active ratio	Overhead (ms) ↓
Flux	Uniform 3×3	9	1.630×	34.14	0.603	1.05
	Uniform 4×4	16	1.610×	34.44	0.606	1.32
	Uniform 6×6	36	1.577×	33.96	0.614	2.94
	Adaptive	–	1.387×	33.68	0.620	518.45
Wan2.1-1.3B	Uniform 7×7	49	1.360×	21.69	0.745	8.46
	Adaptive (coarse)	45	1.333×	22.35	0.752	145.20
	Uniform 14×15	210	1.335×	21.61	0.746	15.12
	Adaptive (fine)	207	1.312×	21.70	0.746	272.53

On Flux, neighboring uniform grids have similar active ratios and quality, while finer or adaptive partitions add overhead without a consistent quality gain. On Wan, coarse adaptive partitioning improves PSNR by 0.66 dB over the matched uniform grid, but this advantage disappears at the fine granularity while the overhead grows. Since the Wan comparison uses a single sample, we present it as evidence of the mechanism rather than a population-level result. Block granularity is therefore a model-dependent trade-off between spatial locality and system overhead, which motivates uniform grids for Flux and coarse adaptive partitions for Wan.

F Visualization of CUBIC

We provide visualization comparison between original video generation and CUBIC on Flux-1, FireRed and Wan 2.1. We evaluate CUBIC on Text-to-Image generation, Image Editing, and Text-to-Video generation using the default resolution of each model. Results demonstrates CUBIC can preserve high pixel-level fidelity, achieving similar generation quality compared with the original generation.

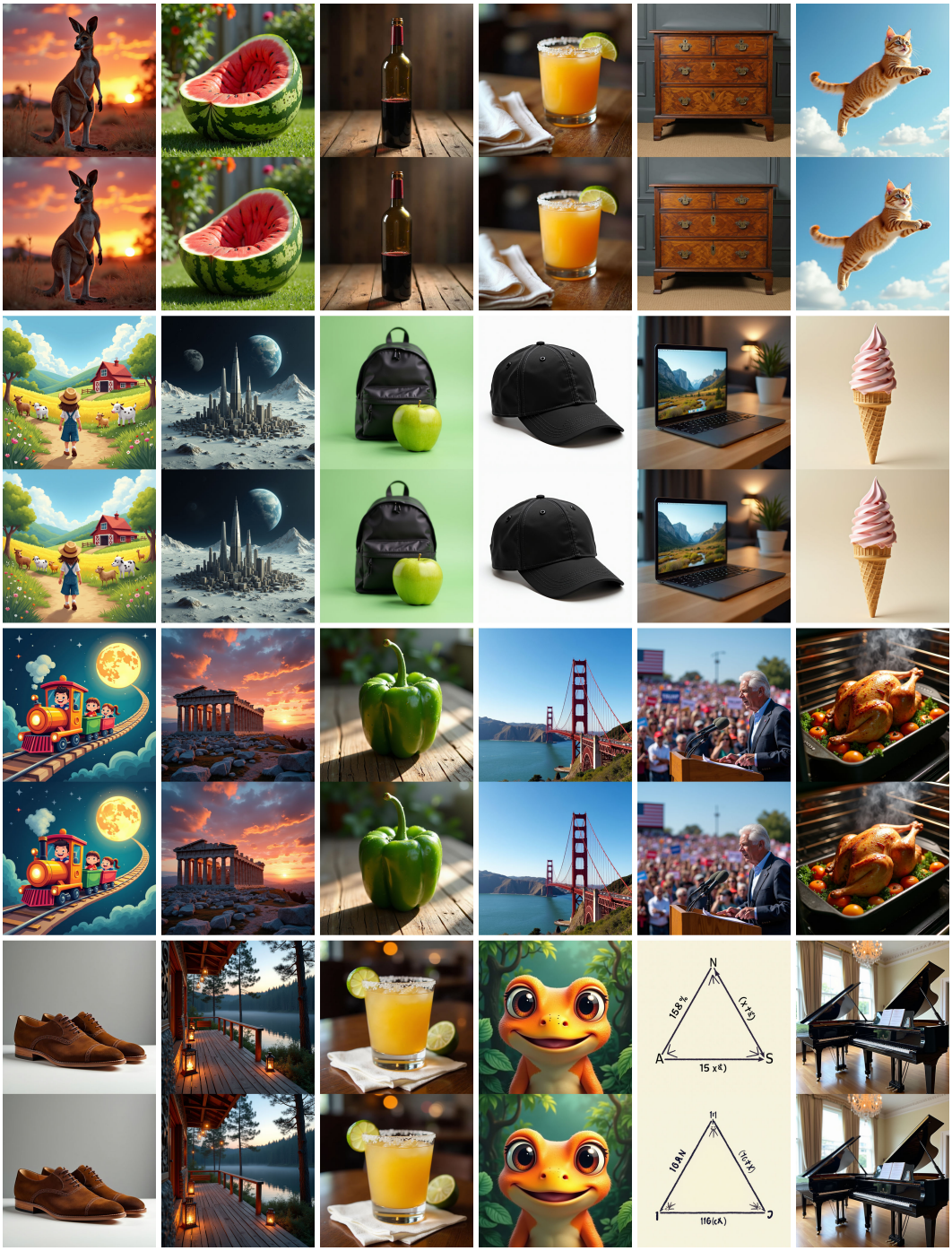

Figure 10: Flux visualization comparison between original images (top) and CUBIC results (bottom)

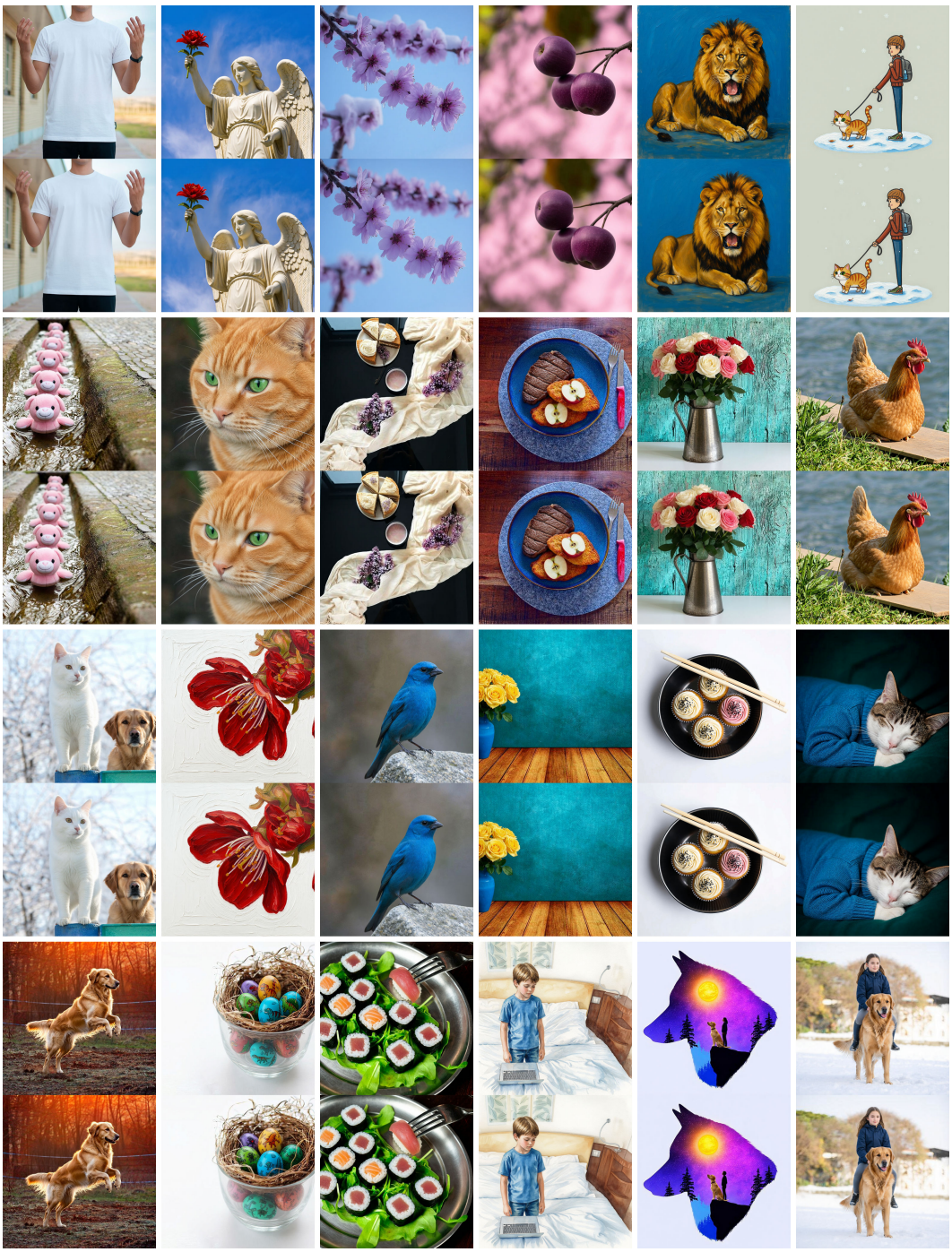

Figure 11: FireRed visualization comparison between original images (top) and CUBIC results (bottom)

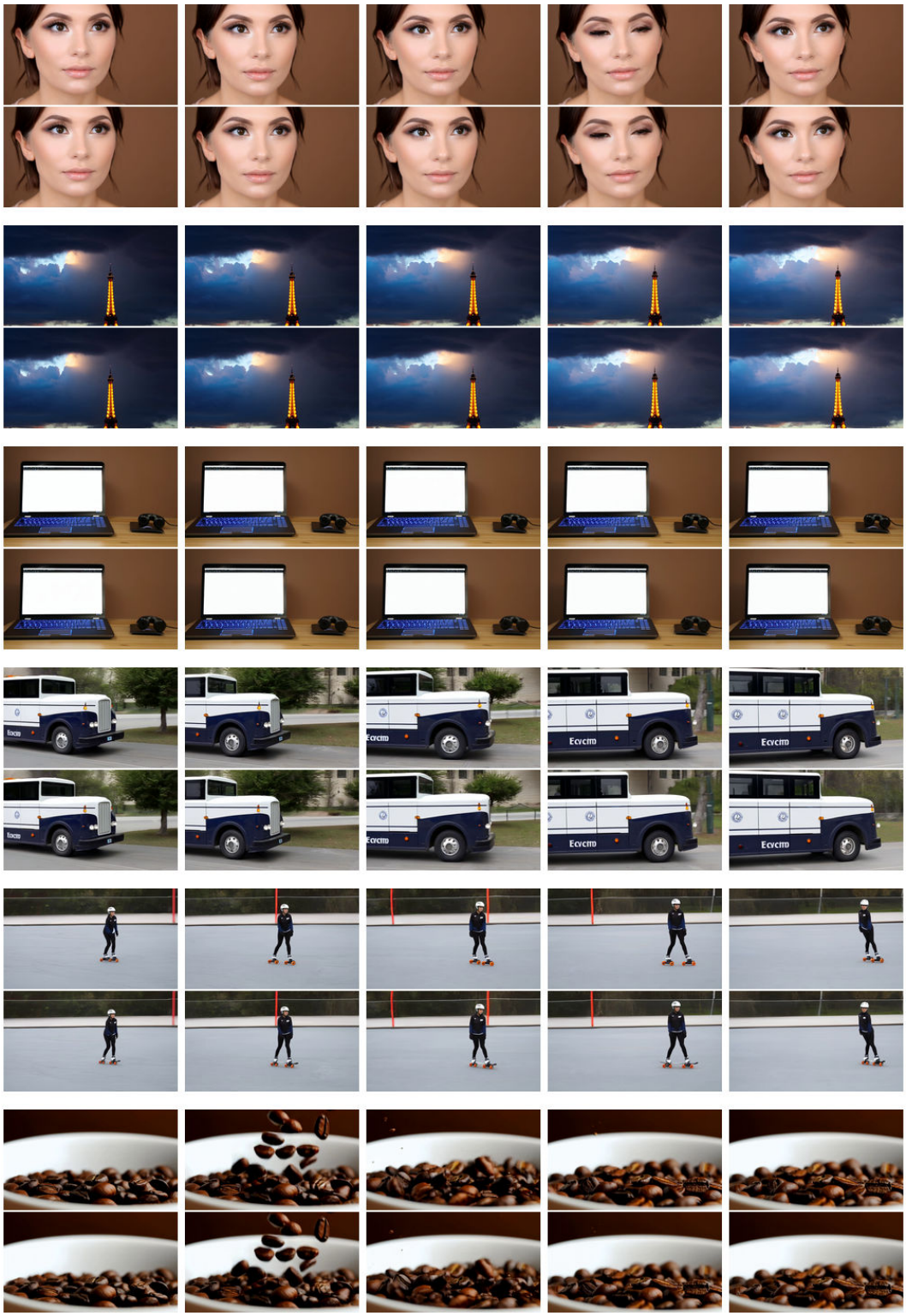

Figure 12: Wan 1.3B visualization comparison between original videos (top) and CUBIC results (bottom)

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Abstract and section 1 reflect the paper's contributions and scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Sec. 6 discuss the limitations of CUBIC.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: Section 4 and Appendix A provide rigorous theoretical derivations and proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide a detailed implementation of CUBIC. All experiments use publicly available datasets and models, ensuring reproducibility.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The implementation, baseline adaptations, configurations, and run scripts are available at `https://github.com/chen-yy20/Cubic`. All models and datasets used are publicly available.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4 and Appendix D outlines the experimental setup used for all results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report the mean and standard deviation of inference time in Table 1. For the comparison with Jano, Appendix E.1 reports per-sample means and standard deviations, paired win counts, and paired Wilcoxon signed-rank tests on held-out prompts.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We present the resources used for experiments in section 5.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This paper conforms with the NeurIPS code of ethics in every respect.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This paper studies a training-free caching strategy for accelerating Flow Matching inference. The method does not introduce new data, new model capabilities, or application-specific deployment scenarios; it only modifies the inference computation schedule to reduce latency. Therefore, potential societal impacts are indirect and primarily depend on the underlying models and their use cases rather than on the proposed acceleration technique itself.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper does not introduce or release new high-risk datasets or generative models.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All the models and datasets are properly cited.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: This paper does not introduce or release new assets such as datasets, pretrained models, or benchmark suites.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: Paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: Paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: CUBIC does not involve LLMs

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.