CineFlow: Breaking Monolithic Video Generation with Dependency-Driven Parallel Execution

Anonymous Author(s)

Abstract

Video diffusion models are rapidly becoming a foundation for generative video applications, but scaling them to long-form generation remains challenging. Preserving visual consistency requires global spatiotemporal attention across frames throughout the diffusion process, leading existing systems to execute generation as a monolithic workload. This design makes even short clips take minutes to generate, causes cost to grow superlinearly with video length, and fundamentally limits parallelism and scalability.

We present CineFlow, a new inference abstraction that reframes video generation as *dependency-driven execution*. The key insight is that global token interaction need not imply global execution dependence: prompt semantics induce a sparse dependency structure over scenes, entities, and events that can expose new parallelism. CineFlow compiles this structure into a dependency graph, executes independent subtasks in parallel, and enforces only the dependencies needed for consistency. However, dependencies are both implicit and dynamic. To address this, CineFlow adopts a conservative-to-adaptive systems design: it first over-approximates dependencies to avoid consistency-breaking parallelism and then refines and prunes using runtime trajectory signals. Across real-world workloads, CineFlow accelerates video generation by 1.7–5.5× over monolithic execution while preserving video generation quality.

ACM Reference Format:

Anonymous Author(s). 2026. CineFlow: Breaking Monolithic Video Generation with Dependency-Driven Parallel Execution. In . ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Recent advances in video diffusion models, such as Wan2.2 [28], CogVideoX [34], and Seedance [9], are rapidly turning generative video into a practical substrate for streaming content creation [11, 13], long-form media synthesis, and filmmaking [43]. Despite rapid progress in model capability, practical deployment remains bottlenecked by inference inefficiency: generating a 5-second clip can take up to 16 minutes [30], and cost scales superlinearly with total frame length due to the quadratic computation complexity of the attention layer. As applications demand longer duration, higher resolution, and richer scene structure, inference efficiency becomes not merely an optimization problem, but a first-order systems challenge [21, 40].

At the heart of this inefficiency lies a fundamental tension: *video diffusion needs global token interaction to preserve spatiotemporal consistency, but efficient inference requires execution units that can be decomposed and parallelized.* Unlike large language model inference, where autoregressive generation naturally exposes execution boundaries through prefill–decode disaggregation and KV caching [15, 44], diffusion transformers (DiTs) repeatedly denoise the full spatiotemporal token sequence across dozens of timesteps using bidirectional attention at each step [22]. This computation pattern keeps frames coupled throughout the diffusion trajectory. Consequently, systems such as xDiT [7] execute video generation as a single monolithic workload, leaving parallelism largely confined to low-level tensor, sequence, or pipeline parallelism and keeping inference constrained by quadratic attention cost as video length or resolution increases (§2.2).

Recent efforts reduce this cost by sparsifying attention, including training-free sparse attention [30] and trained sliding-window/causal variants [31, 33]. The former leaves monolithic execution and its quadratic scaling cost intact; the latter breaks the global bidirectional interactions behind long-range consistency, causing error accumulation [11, 33] and requiring retraining or fine-tuning. Neither changes the underlying systems abstraction: generation remains monolithic and execution dependencies are dictated by the full token sequence.

Complementary to existing advances, we explore a different execution paradigm: *does global token interaction require global execution dependence?* Our key insight is that token-level dependence and execution-level dependence need not be equivalent. Although diffusion models compute over all spatiotemporal tokens, real video prompts exhibit a much sparser dependency structure over scenes, entities, and events. Our analysis of real-world workloads [23] shows that over 68% of prompts contain structured dependencies, such as flashbacks and recurring characters (§2.2). This creates an opportunity to elevate dependencies from tokens to semantic units: preserve coupled diffusion computation within each unit, but coordinate only along the semantic dependency paths needed for visual consistency.

We present CineFlow, a new execution abstraction for video diffusion inference called *semantic dependency decomposition*. Instead of treating video generation as a monolithic sequence, CineFlow compiles a prompt into a directed acyclic graph (DAG) of semantically dependent generation subtasks. Nodes represent schedulable generation units (e.g.,

scenes), while edges capture the dependencies required to preserve visual consistency. This abstraction allows independent regions of the video to be generated in parallel, while consistency-critical dependencies are explicitly enforced.

Realizing this abstraction introduces new challenges at the intersection of serving efficiency and quality (§3). First, dependency extraction is not simple prompt chunking. Free-form prompts rarely state dependencies explicitly, and global constraints such as identity, style, and lighting may be distributed across the prompt. A naive decomposition that treats scenes independently can break visual consistency. Second, dependencies are not static. During denoising, subtasks that initially require coordination may diverge into distinct visual trajectories later, while others may continue to require alignment. Third, dependency-driven execution transforms video generation into a DAG scheduling problem with heterogeneous task durations, dynamic dependencies, and computation–communication trade-offs. Maximizing parallelism therefore requires carefully balancing critical-path latency against consistency preservation.

CineFlow addresses these challenges through a unified *conservative-to-adaptive* systems design (§3). Rather than aggressively exposing parallelism upfront, CineFlow initially over-approximates dependencies to preserve global consistency. As generation progresses, runtime trajectory signals (e.g., changes in latent scenes) provide increasingly accurate evidence about which inter-task dependencies remain important. CineFlow therefore progressively relaxes and prunes dependency enforcement, allowing subtasks to transition from sequential execution to independent runs.

CineFlow introduces three runtime components: (1) *Semantic-Dependency Compiler*: it decomposes a flat, free-form prompt into semantically coherent subtasks and compiles a weighted dependency DAG by jointly reasoning about semantic similarity and causal relationships, exposing new parallelism (§4.1). (2) *Trajectory-Aware Fuser*: To maintain consistency without serializing execution, it streams intermediate latent model features from upstream tasks into downstream denoising, and adaptively prunes these interactions (dependencies) as trajectories diverge. This maximizes execution parallelism and reduces communication overhead (§4.2). (3) *Dependency-Aware Scheduler*: In executing the evolving DAG, it adaptively arbitrates between blocking synchronization (e.g., executing and then waiting for upstream latent updates) and non-blocking counterpart (e.g., buffering and then fetching the update with additional I/O overhead). It maximizes hardware utilization with task backfilling while minimizing overall generation latency with critical path-aware task scheduling (§4.3).

We implement CineFlow atop xDiT [7], preserving compatibility with existing video generation stack (§5). Our evaluations across diverse models, including Wan2.2 [28], CogVideoX [34] and HunyuanVideo [14], and real-world workloads spanning different generation lengths, show that

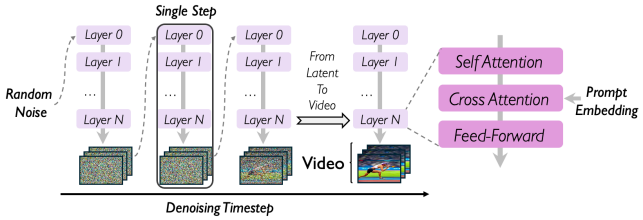


Figure 1: ***The DiT workflow for video generation.***

CineFlow speeds up end-to-end generation by 1.7–5.5× over existing advances (e.g., xDiT [7] and USP [8]) while preserving generation quality (§6).

Overall, this paper makes the following contributions:

- We introduce *semantic dependency decomposition*, a new execution abstraction that elevates video generation from monolithic token-level execution to dependency-driven execution paradigm (§2–§3).
- We propose a *conservative-to-adaptive* runtime that progressively refines dependency enforcement using runtime diffusion signals, enabling new parallelism without sacrificing consistency (§4–§5).
- We build an inference system integrating semantic compiling, trajectory-aware fusion, and dependency-aware scheduling, demonstrating substantial end-to-end speedup while preserving generation quality (§6).

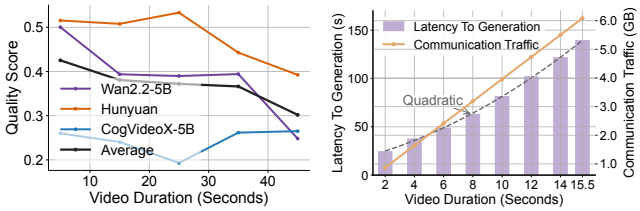
2 Background and Motivation

2.1 Video Diffusion Inference

Video diffusion models, particularly Diffusion Transformers (DiTs) [22], generate videos by iteratively reversing a noise corruption process. As shown in Figure 1, generation starts from Gaussian noise and proceeds through T denoising steps (often $T \approx 50$), each consisting of a full forward pass through a deep transformer. At every step, the model predicts noise conditioned on the current latent state and input prompt, making end-to-end latency proportional to both the number of steps and the cost of each forward pass.

For a video with F frames at resolution $H \times W$, the model processes $F \cdot H \cdot W/p^2$ tokens, where p is the patch size. Unlike autoregressive LLMs, DiTs apply full bidirectional attention over all spatiotemporal tokens at every denoising step. This design is important for global visual consistency, but it also induces a monolithic execution structure: all tokens remain interdependent throughout inference. As a result, computational demands scale quadratically with sequence length, $O((FHW)^2)$, making long-form or high-resolution generation prohibitively expensive.

Monolithic execution creates a quality-structure mismatch. Long videos are not simply longer token sequences; they are structured compositions of scenes, entities, actions,



(a) Quality degradation. (b) Quadratic load increase.

Figure 2: ***Monolithic video generation becomes superlinearly expensive as total frame length increases.***

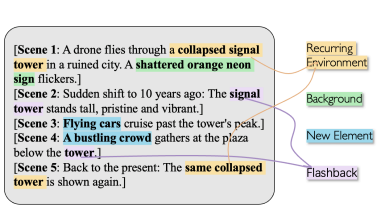


Figure 3: ***Video prompts often induce structured dependencies.***

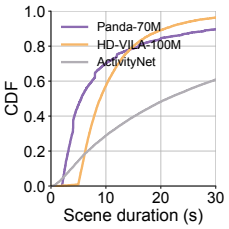


Figure 4: ***Scene durations vary widely.***

transitions, and visual constraints. Current inference systems nevertheless execute them as one dense spatiotemporal rollout. This mismatch becomes increasingly harmful as duration grows. As shown in Figure 2a, state-of-the-art models can produce high-quality short clips but degrade over longer horizons, exhibiting temporal inconsistency and semantic drift. In some cases, quality drops align with implicit scene boundaries, suggesting that the model is forced to maintain global interaction even when the content naturally contains separable semantic units.

This observation points to a deeper systems issue. Monolithic execution treats every frame as equally dependent on every other frame throughout the entire diffusion trajectory. It becomes overly conservative for structured narratives that contain scene cuts, viewpoint changes, or weakly related events. By enforcing dense interaction across the entire video, existing systems suppress decomposable structure that could otherwise expose parallelism and improve scalability.

Quadratic scaling cost limits system efficiency. The monolithic execution structure also creates severe system bottlenecks. Increasing video length or resolution rapidly amplifies compute, memory, and communication demand. As shown in Figure 2b, doubling sequence length leads to nearly 4× higher latency, quickly exceeding the capacity of modern GPUs in ensuring responsiveness.

Existing systems such as xDiT [7] and DistriFusion [17] distribute computation across GPUs using tensor, sequence, or pipeline parallelism [26]. However, these approaches preserve the same monolithic execution abstraction: (i) they do not reduce total computation, only partition it; and (ii) they require frequent all-to-all communication at every denoising step to maintain global attention. As sequence length grows, this communication overhead quickly dominates execution time, leading to diminishing returns from scaling [8].

2.2 Opportunities for Decomposing Monolithic Generation

We challenge the prevailing assumption that video diffusion needs to be executed as a single monolithic workload. Instead, we argue that long-form video generation can be reformulated as *dependency-driven execution*: a graph of semantic generation units (subtasks) whose dependencies are enforced only when necessary. This reformulation can expose parallelism, reduce global synchronization, and preserve quality without modifying or retraining the underlying model. Our motivation comes from two observations.

Prompts induce structured generation dependencies. Video prompts are rarely flat frame-by-frame descriptions. They encode rich narrative structure: scenes, entities, and temporal dependencies. These relationships often form a dependency graph rather than a simple sequence. For example, a prompt may describe a character introduced in one scene, multiple subsequent scenes that share that character or style, and a later scene that depends on the outcome of an earlier event. Such structure naturally gives rise to patterns such as *Linear* dependencies for sequential progression, *Expansion* dependencies for one-to-many branching, *Convergence* dependencies for many-to-one merging, and *Recurrence* dependencies for long-range lookbacks.

To quantify this structure, we analyze the MovieGenBench dataset [23] and categorize over 1K prompts using Gemini as a judge [41]. As shown in Figure 5, more than 68% of real-world prompts exhibit non-linear dependencies. Importantly, these dependencies are selective (Figure 3): some scenes must remain strongly coordinated because they share entities, causal state, or visual style, while other scenes are weakly related and can be generated with little cross-scene interaction.

This dependency sparsity is further amplified by temporal heterogeneity. Figure 4 shows that scene durations vary widely, with many segments extending beyond 20 seconds. This heterogeneity creates load imbalance under naive decomposition, but also exposes scheduling flexibility: independent long and short segments can be overlapped if the system understands their dependency structure.

> **Implication 1:** Monolithic video generation is overly conservative: it enforces global dependencies across all tokens even when the generation dependency needed for consistency is often sparse.

Dependency requirements change during denoising. Beyond prompt structure, the diffusion process itself is not

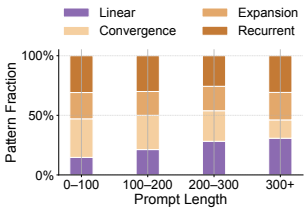


Figure 5: ***Dependency patterns in prompt narratives.***

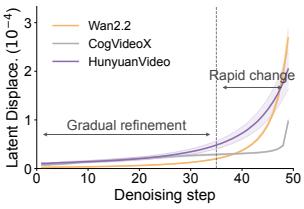


Figure 6: ***Latent displacement changes over steps.***

uniformly sensitive to dependencies over time. As shown in Figure 6, latent updates exhibit distinct phases. Early denoising steps (e.g., first 30 steps) produce relatively small updates and establish coarse global structure aligned with the prompt. Later steps increasingly refine scene-specific details such as texture, motion, and local appearance, causing latent trajectories to diverge.

This temporal heterogeneity implies that dependency requirements are *phase-dependent*: early stages benefit from strong coordination to establish global consistency, while later stages can execute independently for local refinement.

> **Implication 2:** Dependency is not static: it should be enforced conservatively early, and can be relaxed as generation releases more semantic signals.

Challenges toward structured and dynamics-aware execution. These observations reveal an untapped systems design space. Long-form video generation can be represented as a *structured DAG of semantic subtasks* with *time-varying dependencies*. This representation exposes parallelism across prompt structure, while preserving consistency through selective dependency enforcement. However, realizing this execution model requires solving three unique challenges:

- *Semantic Compiling Challenge*: Prompts are free-form and ambiguous, yet systems execution requires concrete task boundaries, dependency directions and strengths. The system must infer a DAG that is expressive enough to expose parallelism yet conservative enough to avoid breaking visual consistency.
- *Runtime Consistency Challenge*: Monolithic execution preserves consistency through global attention. Once generation is decomposed, the system must explicitly propagate sufficient latent information across dependent subtasks, without large synchronization overhead.
- *Scheduling Challenge*: A DAG with heterogeneous task durations and dynamics creates a complex scheduling problem (e.g., stragglers and critical paths). Moreover, the system must carefully balance parallelism and dependency enforcement to preserve generation quality.

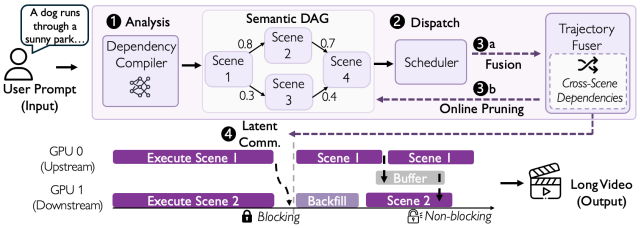


Figure 7: ***CineFlow overview and execution workflow.***

3 System Overview

We present CineFlow, the first video generation system that transforms video generation from monolithic execution into a structured, dependency-driven workflow. The key design principle of CineFlow is *conservative-to-adaptive execution*: dependencies are first compiled conservatively from prompts and then progressively refined using runtime signals.

Given a prompt, CineFlow compiles the generation process into a DAG, $\mathcal{G} = (\mathcal{S}, \mathcal{E})$, where each node $s_i \in \mathcal{S}$ represents a semantically coherent segment, and each edge (s_i, s_j) encodes a dependency required for visual consistency. This separates the *logical dependency structure* of a video from its *physical execution schedule*, allowing CineFlow to expose parallelism without discarding the coordination needed for coherent generation.

System Workflow. As shown in Figure 7, when a request arrives, ❶ the *Semantic-Dependency Compiler* parses the input prompt into semantically coherent tasks, identifies task boundaries (e.g., scenes), infers dependency directions, and assigns dependency weights that capture the strength of required coordination (§4.1). The compiler intentionally over-approximates dependencies to avoid spurious independence before generation begins. ❷ The *Dependency-Aware Scheduler* maps DAG nodes to GPUs and launches tasks when their dependencies permit (§4.3). It treats semantic tasks as the unit of execution and optimizes for execution parallelism, critical-path latency, hardware utilization, and synchronization cost. ❸ During execution, the *Trajectory-Aware Fuser* enforces cross-task dependencies to ensure global consistency, by propagating latent updates from upstream to downstream tasks along DAG edges. With these runtime signals, it performs *online pruning* on the DAG, thereby stopping latent communication once the execution unit enters an independent regime (§4.2). ❹ For each dependency at runtime, the scheduler chooses between blocking synchronization and asynchronous buffered exchange, balancing latency, communication overhead, and GPU utilization. Finally, execution units are merged into a coherent video.

4 CineFlow Design

We now describe how CineFlow realizes dependency-driven video generation as a runtime system via: (1) *Semantic-Dependency Compiler*: compiles a free-form prompt into

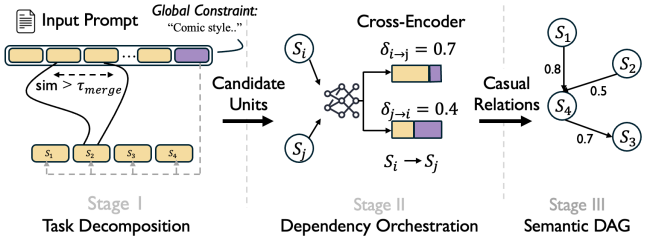
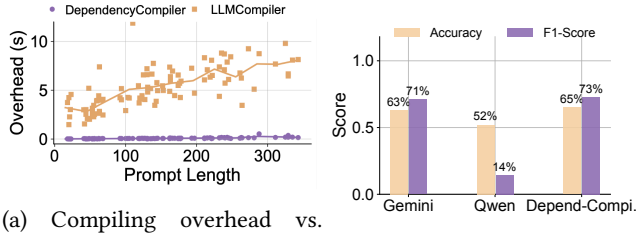


Figure 8: ***Dependency compiler converts unstructured prompts into a dependency DAG via: (I) Decomposition: Segments the prompt and merges close fragments. (II) Orchestration: Resolves causal dependency between tasks. (III) Compilation: Synthesizes a weighted, acyclic DAG.***



(a) Compiling overhead vs. length.

(b) Reorder accuracy.

Figure 9: ***Our compiler enables smaller overhead, higher accuracy, and dense signals for later online adaptation.***

an execution DAG (§4.1); (2) *Trajectory-Aware Fuser*: mitigating the intensive communication overhead in cross-task dependencies (§4.2), and (3) *Dependency-Aware Scheduler*: orchestrating tasks to optimize critical-path latency (§4.3).

4.1 Semantic-Dependency Compiler: Build DAGs

The Dependency Compiler constructs the execution graph on which CineFlow operates. Its goal is not to perfectly capture every semantic relation of a prompt, but rather to compile a conservative dependency DAG that exposes parallelism without introducing spurious independence. The resulting DAG provides the execution substrate for trajectory fusion and runtime scheduling.

Designing such a compiler is challenging for three reasons. First, semantic dependencies are not necessarily aligned with surface prompt order (§2.2): adjacent fragments may describe independent scenes, while distant fragments may share entities, visual style, or causal state. Second, dependency errors are asymmetric. Missing a dependency can cause irreversible visual inconsistency, whereas adding an unnecessary dependency only reduces parallelism and can later be pruned. Third, some dependencies cannot be determined reliably before generation begins, because the need for coordination depends on how latent trajectories evolve during denoising. CineFlow therefore adopts a conservative compilation strategy: it constructs a dependency-rich graph *before execution* and delegates dependency relaxation to the runtime fuser (§4.2).

Stage I: Semantic Task Decomposition. The first stage identifies generation units that are large enough to remain visually coherent, yet small enough to expose parallelism. Simple syntactic splitting, such as by punctuation or sentence boundaries, is insufficient: it can separate fragments that share temporal continuity or recurring entities, while failing to separate independent scenes described in adjacent text. Conversely, overly fine-grained decomposition can break visual coherence by forcing the runtime to coordinate too many small fragments.

CineFlow formulates generation decomposition as a bottom-up merging process (Figure 8). It first over-segments the prompt into candidate fragments and encodes each fragment into a dense embedding space using a lightweight text encoder (e.g., MiniLM [29]). Adjacent fragments are then merged if their similarity (e.g., cosine similarity of dense embeddings) exceeds a threshold τ_{merge}, yielding segments $\{s_1, \ldots, s_n\}$ that correspond to maximally coherent visual units. This process adapts naturally to prompt structure: complex prompts yield more generation tasks, while tightly dependent descriptions remain grouped into a single task.

Beyond local coherence, prompts often contain global constraints (e.g., style, lighting, or recurring attributes) that span multiple segments. CineFlow extracts such signals using lightweight keyword- and embedding-based aggregation [6, 25], and attaches them as shared conditioning context to those segments.

Stage II: Directional Dependency Graph Compilation. Decomposition determines what units to generate, but not how those units depend on each other. Textual order alone is unreliable: prompts may contain flashbacks, recurring entities, branching events, or descriptions whose causal order differs from their surface order (Figure 3).

CineFlow casts dependency compilation as a pairwise entailment problem [2, 25]. For each segment pair (s_i, s_j), the compiler leverages the lightweight entailment model (e.g., RoBERTa [20] or MiniLM [29]) to estimate whether s_i provides semantic or temporal context for s_j. The model produces entailment and contradiction probabilities, P_{entail} and P_{contra}. We define the directional dependency score $\delta(i, j) = P(\text{entail} \mid s_i, s_j) - P(\text{contra} \mid s_i, s_j)$.

A high value of $\delta(i, j)$ indicates that s_i is likely to constrain or precede s_j, while a high contradiction score reduces the dependency strength. Because the score is directional, $\delta(i, j)$ and $\delta(j, i)$ can differ, allowing the compiler to capture asymmetric dependencies.

The compiler next converts pairwise scores into an executable DAG. For each segment pair (s_i, s_j), it compares the two directional scores and inserts an edge from s_i to s_j when $\delta(i, j) > \delta(j, i)$. Edge weights record the dependency strength and are later used by the fuser to determine fusion

strength and pruning priority. If the resulting graph contains cycles, the compiler removes edges with the smallest confidence margins until the graph becomes acyclic, preserving the strongest directional signals.

The compiled DAG is intentionally conservative. Rather than discarding low-confidence dependencies before generation, CineFlow retains them when they may affect consistency and allows the trajectory-aware fuser to prune them at runtime (§4.2). This design reflects the asymmetric cost of errors: an extra edge only reduces parallelism temporarily, but a missing edge can hurt generation quality.

Efficiency Analysis. The compiler runs in $O(s^2)$ time over s segments, which remains negligible as $s \ll N$ (token count). More importantly, the resulting decomposition enables dependency-driven execution. Partitioning a sequence of token generation length N into K segments reduces the model's computation cost from $O(N^2)$ to $O((N/K)^2 \times K)$, enabling linear speedup in theory.

Figure 9 compares our compiler against LLM-based dependency compilation. Our design incurs negligible latency (< 0.1s), whereas the LLM-based approach already introduces seconds of overhead for moderately long prompts. Moreover, our compiler achieves higher dependency inference accuracy while producing quantitative dependency strengths required by downstream DAG pruning and trajectory fusion (§4.2). Detailed evaluation is available in Appendix D.

4.2 Trajectory Fuser: Align Inter-Task Generation

The semantic-dependency DAG exposes parallelism by conservatively decomposing generation into tasks before generation starts. However, this conservatism inevitably includes dependencies that are not ultimately needed, because before generation starts the system lacks runtime evidence about which subtasks will remain visually dependent. The key runtime question is how to preserve cross-task consistency while progressively removing unnecessary synchronization.

This question becomes challenging because visual coherence in diffusion models is established through the evolution of latent trajectories. Small early deviations between dependent tasks can amplify over denoising steps and appear as identity drift, style mismatch, or broken causal continuity in downstream segments. Yet, enforcing all dependencies throughout the full trajectory would collapse the exposed parallelism and too frequent synchronization can even harm quality by making distinct segments overly similar, as shown in Figure 10. Thus, CineFlow must synchronize just enough latent state to preserve shared structure, but stop synchronizing once tasks enter independent refinement.

CineFlow addresses this with a *Trajectory-Aware Latent Synchronization* mechanism. Its design follows two observations from §2.2: (1) early alignment is critical, as global structure is established in the high-noise regime; and (2) late

Figure 10: ***Full-step fusion leads to identical clips, losing unique identities (e.g., Evening vs. Night).***

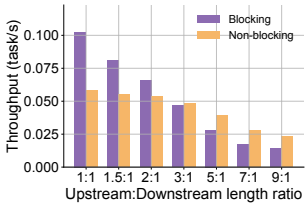


Figure 11: ***Impact of workload skew on communication efficiency.***

independence is desirable, as trajectories naturally diverge to form distinct content. Accordingly, the fuser operates in two stages: (i) *inter-task trajectory fusion*, which propagates upstream latent updates to downstream tasks along active DAG edges; and (ii) *trajectory-aware pruning*, which disables synchronization once runtime signals indicate that a dependency is no longer needed.

Stage I: Inter-Task Trajectory Fusion. To coordinate dependent tasks without waiting for entire upstream tasks to finish, CineFlow exposes synchronization points (i.e., time to merge the attention map) at the granularity of denoising steps and transformer layers. This converts each task into a sequence of micro-tasks and allows a downstream task to start early while progressively incorporating upstream updates. The result is a streaming dependency interface: upstream tasks publish intermediate latent features, and downstream tasks consume them only along active DAG edges.

The core operation is inter-task latent fusion. Let $\mathbf{h}_{up}$ denote the upstream latent feature and $\mathbf{h}_{down}$ denote the downstream latent feature at a synchronization point. Directly injecting upstream features, such as $\mathbf{h}'_{down} = \mathbf{h}_{down} + \mathbf{h}_{up}$, can shift the downstream feature distribution too close to the upstream, leading to overly similar generation (Figure 10). CineFlow therefore normalizes the upstream feature and rescales it to match the downstream statistics: $\mathbf{h}_{up}^{norm} = \frac{\mathbf{h}_{up} - \mu(\mathbf{h}_{up})}{\sigma(\mathbf{h}_{up}) + \epsilon}$, where $\mu(\cdot)$ and $\sigma(\cdot)$ compute the mean and variance across the spatial dimensions. Then the fuser incorporates upstream latent update to the downstream by:

$$\mathbf{h}'_{down} = (1-\alpha)\mathbf{h}_{down} + \alpha \cdot \left(\mathbf{h}_{up}^{norm}\sigma(\mathbf{h}_{down}) + \mu(\mathbf{h}_{down})\right) \quad (1)$$

where $\alpha \in [0, 1]$ controls the fusion strength. This makes upstream information act as a distribution-aligned prior rather than a hard overwrite, while injecting shared structure.

For a downstream task with multiple predecessors, CineFlow combines upstream features according to their dependency weights. Given predecessors $s_{up,i}$, the contribution from $s_{up,i}$ is weighted by $w_{i\to down} = \frac{\delta(s_{up,i}, s_{down})}{\sum_j \delta(s_{up,j}, s_{down})}$, ensuring that dominant predecessors contribute proportionally.

Stage II: Task Dependency Pruning. Fusion is useful for establishing consistent structure, but it should not persist longer than necessary. Early denoising steps determine

coarse layout, identity, style, and causal continuity; later steps increasingly specialize each task with local texture, motion, and appearance. Continuing to synchronize during this local-refinement phase increases communication overhead and can over-constrain generation, causing distinct segments to converge toward similar outputs (Figure 10).

CineFlow therefore treats dependencies as runtime state rather than fixed graph edges. For each active edge, the fuser tracks the alignment between upstream and downstream latent trajectories. Once the divergence of $\mathbf{h}_{up}$ and $\mathbf{h}_{down}$) exceeds threshold τ, the runtime disables fusion (communication) on that edge for the remaining denoising steps. This allows downstream tasks to transition to fully independent execution, unlocking parallelism while preserving already-established structure. Our studies show that CineFlow achieves consistently superior performance over a wide range of τ (§6.4).

We next show that CineFlow introduces a bounded quality deviation from existing monolithic diffusion execution.

Theorem 4.1 (Bounded gap in trajectory fusion). *Let $x^M(t)$ be the latent trajectory under monolithic diffusion and $\hat{x}(t)$ be the trajectory under CineFlow. Assume the denoising operator F is L-Lipschitz [4, 16, 27], then:*

$$\left|\hat{x}(T) - x^M(T)\right| \leq \int_0^T e^{L(T-s)} \cdot \left(\bar{\varepsilon}_{\text{struct}}(s) + \bar{\varepsilon}_{\text{fuse}}(s)\right) ds.$$

The structural error $\bar{\varepsilon}_{\text{struct}}$ captures approximation introduced by dependency decomposition and adaptive dependency pruning, while the fusion error $\bar{\varepsilon}_{\text{fuse}}$ captures the mismatch between monolithic latent interaction and CineFlow's trajectory fusion. As these errors decrease, CineFlow's trajectory remains close to the monolithic trajectory. We provide the full proof in Appendix A.

4.3 Runtime Scheduler: Minimize Latency

While the compiler exposes parallelism and the fuser refines dependencies, end-to-end speedup ultimately hinges on how tasks are scheduled. The key challenge is to minimize critical-path latency under heterogeneous task durations, evolving dependencies, and different synchronization costs. Fortunately, the computation duration of diffusion workloads is highly predictable and content-agnostic, depending on token counts which are known in advance from the user-specified frames to generate (§2.1). This enables a cost-aware scheduler that explicitly reasons about computation–communication trade-offs and proactively reshapes the execution timeline.

Scheduling Computation–Communication Overlap. Given the DAG $\mathcal{G} = (\mathcal{S}, \mathcal{E})$, each task $s_i \in \mathcal{S}$ is decomposed into micro-tasks $\mathcal{M}_i = \{m_{i,t,k}\}$ across denoising steps t and transformer layers k. An edge (s_{up}, s_{down}) means that s_{down} consumes latent features produced by s_{up}. As shown in Figure 12, trajectory fusion introduces *schedulable overlap*

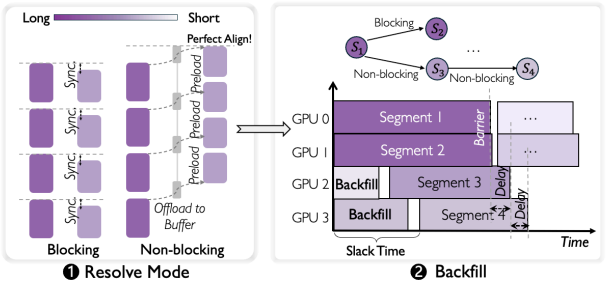


Figure 12: ***CineFlow's Predict-and-Backfill Scheduler (1) resolves communication modes via cost-aware modeling and (2) backfills asynchronous slack windows with other ready tasks to minimize generation makespan.***

between computation and communication, giving rise to two synchronization structures:

- *Blocking synchronization.* The upstream and downstream tasks progress together and synchronize at each micro-task boundary, such as each transformer layer within each denoising step. This avoids materializing long-lived latent buffers, but it introduces repeated execution bubbles: the downstream task cannot advance until the corresponding upstream latent feature is available. We model the per-boundary overhead as $w_{\text{blk}} = |t_{up} - t_{down}| + t_{\text{g2g}}$, where $|t_{up} - t_{down}|$ captures idle time caused by progress skew between dependent tasks, and t_{g2g} captures GPU–GPU transfer time. Because this overhead recurs across steps and layers, blocking synchronization is efficient only when dependent tasks run at similar speeds.
- *Non-blocking buffered synchronization.* The upstream task materializes intermediate latent features into a buffer (GPU or CPU memory), allowing the downstream task to proceed independently and consume latents when needed. This removes repeated lock-step bubbles, but may introduce a dependency delay if the required latent has not yet been produced, as well as I/O overhead when latents must be transferred or offloaded. We model this overhead as $w_{\text{non\_blocking}} = \max\left(0, t^{up}_{\text{ready}} - t^{down}_{\text{start}}\right) + t_{\text{transfer}}$, where t^{up}_{ready} is the time when the required upstream latent becomes available, and t_{transfer} captures the data transfer cost. For example, GPU–CPU buffer offloading is often required due to the large latent size of many frames and over steps—considering even a frame's latent size can be as large as 4 GB. This mode decouples the blocking execution, releasing GPUs for other ready tasks.

Figure 11 shows that the better mode depends on workload skew, with end-to-end generation latency differing by up to 36%. Moreover, the choice cannot be made purely locally: a delay on a task near the root of the DAG may propagate through many downstream tasks, while the same delay on a

Algorithm 1 CineFlow Scheduling Algorithm

Input: DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, GPU pool $\{G_1, \ldots, G_N\}$
Output: Dispatch assignments $\mathcal{D}$ and communication modes $\mathcal{M}$

1: **function** SCHEDULE($\mathcal{G}$)
2: $\mathcal{Q} \leftarrow$ INITQUEUE($\mathcal{G}$) ▷ Descending frame length
3: $\mathcal{D} \leftarrow \emptyset$, $\mathcal{M} \leftarrow \emptyset$, AvailGPU $\leftarrow N$
4: **while** $\mathcal{Q} \neq \emptyset$ **and** avail > 0 **do**
5: $s \leftarrow \mathcal{Q}$.POP()
6: $\mathcal{M}_s \leftarrow$ RESOLVEMODE(s, $\mathcal{D}$)
7: **if** ALLNONBLOCK($\mathcal{M}_s$) **then**
8: $T_{slack,s} \leftarrow$ BACKFILLWINDOW(s, $\mathcal{M}_s, \kappa$)
9: $\mathcal{D} \leftarrow \mathcal{D} \cup$ BACKFILL($\mathcal{Q}$, $T_{slack,s}$)
10: **end if**
11: $\mathcal{D} \leftarrow \mathcal{D} \cup \{s\}$, AvailGPU $\leftarrow$ AvailGPU $- 1$
12: UPDATEQUEUE(s, $\mathcal{Q}$, $\mathcal{D}$)
13: **end while**
14: **return** $\mathcal{D}$, $\mathcal{M}$
15: **end function**

16: **function** RESOLVEMODE(s, $\mathcal{D}$)
17: **for** $u \in$ upstream(s) **do**
18: **if** $u \in$ running **then**
19: $\mathcal{M}_s[u] \leftarrow$ NON-BLOCKING
20: **else** ▷ Compare cumulative overheads
21: $\text{Cost}_{\text{blocking}} \leftarrow w_{\text{blocking}}(u, s) \cdot N_{step} \cdot N_{layer}$
22: $\text{Cost}_{\text{non\_blocking}} \leftarrow w_{\text{non\_blocking}}(u, s) \cdot \text{Depth}(s)$
23: **if** $\text{Cost}_{\text{blocking}} \leq \text{Cost}_{\text{non\_blocking}}$ **then**
24: $\mathcal{M}_s[u] \leftarrow$ BLOCKING
25: **else**
26: $\mathcal{M}_s[u] \leftarrow$ NON-BLOCKING
27: **end if**
28: **end if**
29: **end for**
30: **return** $\mathcal{M}_s$
31: **end function**

32: **function** BACKFILLWINDOW(s, $\mathcal{M}_s$)
33: **return** $\max_{u \in \text{upstream}(s)} |t_u - t_s| \times N_{step} \times N_{layer}$
34: **end function**

leaf may have little impact. We therefore model the generation makespan of a request as:

$$T(\mathcal{G}) = \max_{p \in \text{Paths}(\mathcal{G})} \sum_{s \in p} \Big(T_{\text{comp}}(s) + \hat{T}_{\text{comm}}(s) \Big), \tag{2}$$

where $\hat{T}_{\text{comm}}(s)$ is the non-overlapped communication overhead induced by the selected synchronization mode. The scheduler's goal is to reduce this critical-path latency while keeping GPUs fully utilized for high throughput.

Predict-and-Backfill Scheduling. Jointly optimizing synchronization mode selection and task scheduling to minimize $T(\mathcal{G})$ is NP-hard, as it generalizes DAG scheduling with additional binary decisions per edge [10]. This makes optimal decisions impractical for latency-sensitive inference. Therefore, we next introduce a new *predict-and-backfill* scheduling algorithm and later prove its competitive effectiveness.

The scheduler first projects the critical-path impact of each synchronization mode. Blocking synchronization incurs repeated micro-task stalls at each layer and step, so its cumulative cost is $\text{Cost}_{\text{blocking}} = N_{\text{step}} \cdot N_{\text{layer}} \cdot w_{\text{blocking}}$. For non-blocking buffered execution, the cost manifests as a one-time delay that propagates down to the DAG, i.e., $\text{Cost}_{\text{non\_blocking}} = w_{\text{non\_blocking}} \cdot \text{Depth}(s)$, where $\text{Depth}(s)$ is the length of the longest downstream path from task s.

As in Algorithm 1, CineFlow maintains a queue $\mathcal{Q}$ of ready tasks, prioritized by the frame generation length of each task in a descending order, since longer tasks are more likely to dominate makespan. In each scheduling round, the scheduler selects the longest task s and invokes RESOLVEMODE (Line 16) to assign its synchronization modes with the upstream task. If an upstream u is already active, the scheduler assigns NON-BLOCKING (Line 19) synchronization because the upstream task already generates the latent update and has to buffer it. For other dependencies, the scheduler compares Cost $*$ blk and Cost $*$ nb and greedily chooses the lower-cost mode (Line 23), thereby greedily optimizing critical-path latency.

As shown in Figure 12(b), when the dependency of task s is resolved as non-blocking buffered synchronization (Line 7), the scheduler enters the slack-aware backfilling phase as we can defer this task's execution and repurpose the resulting execution slack. It invokes BACKFILLWINDOW (Line 8) to compute the cumulative slack $T_{\text{slack},s}$, representing the maximum slack by which s can be delayed without stalling its downstream task (e.g., on the non-critical path). The BACKFILL (Line 9) then fills this window with short, ready tasks from $\mathcal{Q}$. This converts potential communication or dependency slack into useful computation, improving GPU utilization without increasing the critical path. After dispatching a task, UPDATEQUEUE enqueues newly eligible downstream tasks and continues scheduling until no ready task or GPUs.

We next prove that our predict-and-backfill scheduling achieves competitive performance guarantee.

Theorem 4.2 (Critical-path approximation)**.** *Let $T^*(\mathcal{G})$ denote the optimal makespan of DAG $\mathcal{G}$ under optimal synchronization mode assignment, and let $T^{CineFlow}(\mathcal{G})$ denote the makespan under our heuristic. Assume: (i) the skew between dependent tasks is bounded, $|t_{up} - t_{down}| \leq \Delta$, (ii) the non-blocking transfer cost is bounded, $t_{\text{transfer}} \leq C$; and (iii) the DAG has finite depth D. Then the additional critical-path delay of CineFlow is bounded by $T^{CineFlow}(\mathcal{G}) - T^*(\mathcal{G}) \leq O\big(D \cdot \min\{\Delta \cdot N_{step} N_{layer}, C\}\big)$.*

Proof sketch. For each dependency edge, blocking synchronization incurs repeated micro-task stalls, upper bounded by $O(\Delta \cdot N_{\text{step}} N_{\text{layer}})$, while non-blocking synchronization incurs a bounded transfer delay $O(C)$. Since CineFlow explicitly compares these two costs and selects the lower-cost mode, the additional delay introduced on each dependency edge is bounded by $O(\min\{\Delta \cdot N_{\text{step}} N_{\text{layer}}, C\})$. Any critical path contains at most D dependency levels, so the cumulative

extra delay over the critical path is bounded by the stated expression. Backfilling can only reduce idle time by filling slack with ready independent tasks, so it does not increase the critical-path upper bound. We empirically validate the effectiveness of this heuristic in §6.4.

5 Implementation

We implemented CineFlow in over 6,000 lines of Python, extending the popular xDiT [7] to provide seamless support for modern video diffusion models. The system consists of a compiler frontend and a distributed runtime.

Software Stack. The Semantic-Dependency Compiler is implemented using the `sentence-transformers` library. We use `all-MiniLML6-v2` [25] for generating sentence embeddings and `nliMiniLM2-L6-H768` [29] as the cross-encoder for NLI-based dependency resolution. Both models run on the host CPU to avoid competing with the generative DiT models for VRAM. For the underlying video generation, we integrate support for Wan2.2, CogVideoX and HunyuanVideo, leveraging their `diffusers` implementations.

Distributed Runtime. The runtime adopts a multi-process architecture with one worker per GPU using Python `multiprocessing`. Communication is implemented with `torch.distributed` (NCCL backend). To support DAG execution, we design a barrier-based dispatcher that maintains globally consistent task states. To avoid communication deadlocks under asymmetric dependencies, we introduce a two-phase protocol: (1) all ranks perform an `all_reduce` over a dependency bitmask, and (2) derive a deterministic schedule of P2P `isend`/`irecv` operations. This ensures correctness while preserving communication-computation overlap.

6 Evaluation

We evaluate CineFlow across state-of-the-art models and real-world workloads. Our results are summarized as follows:

- CineFlow achieves 1.7–5.5× end-to-end speedups and 17.3% higher overall quality scores than state-of-the-art optimizations and systems (§6.2).
- CineFlow transforms video generation from monolithic execution into quality-preserving orchestration, where the scheduler acts as a guardian of generative fidelity and system performance (§6.3).
- CineFlow exhibits strong robustness across diverse sampling steps and scheduler configurations, delivering consistent performance gains as hardware scales (§6.4).

6.1 Methodology

Experiment Setup and Workloads. We conduct experiments on a cluster of 8 NVIDIA H100 GPUs. To demonstrate the generalizability, we evaluate three representative and widely-used models: Wan2.2-5B [28], CogVideoX-5B [34], and HunyuanVideo (around 8B) [14].

We use real-world prompts from MovieGenBench [23], which contains over 1K complex, multi-sentence narrative prompts. Following the methodology in video generation advances [21, 40], we extend prompts with GPT to introduce richer narrative structure and denser constraints. To model realistic temporal heterogeneity, we sample scene durations from ActivityNet [37] and Panda-70M [5] distributions, assigning such realistic duration requirements to each scene.

Baselines. We compare CineFlow against two categories of baselines focusing on efficiency and generation paradigm:

System Efficiency Baselines: These represent the prevailing monolithic execution systems using standard parallelisms:

- *USP [8]*: Sequence parallelism combining All-to-All and ring-based communication for long-sequence attention.
- *Megatron [26]*: Industry-used tensor parallelism frameworks that shards model weights across GPUs.
- *xDiT [7]*: A state-of-the-art DiT inference engine, using a hybrid of USP and TP to maximize efficiency.

Generation Paradigm Baselines: These represent alternative strategies for long-form video synthesis:

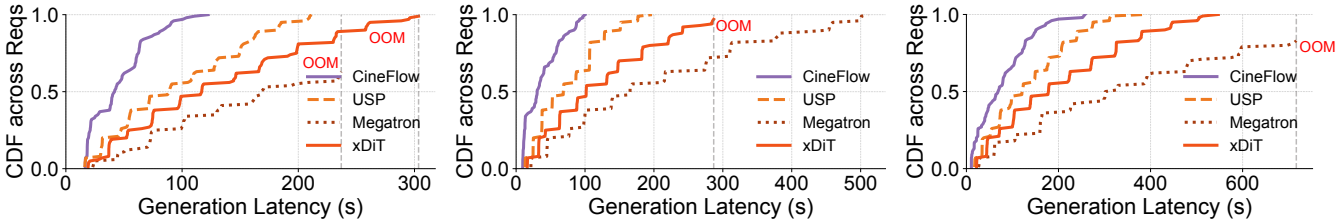
- *Monolithic*: Standard full-sequence execution that treats the multi-scene prompt as a single continuous sequence.
- *LongLive [33]*: An advanced video generation framework for long-form video generation with temporal sliding windows in attentions.
- *Mask2DiT [24]*: Independent scene generation followed by concatenation, with zero dependency.

Evaluation Metrics. Practical video generation deployments care about two dimensions:

- *Serving Efficiency*: Generation latency and serving throughput in terms of average Frames per Second (FPS).
- *Visual Fidelity and Consistency.* Measured using the popular VBench [12] suite that reports video generation quality, focusing on five key metrics: subject consistency and background consistency to measure the consistency across scene boundaries; motion smoothness to ensure the fluidity of the generated content; imaging and aesthetic quality to quantify the high-fidelity visual output.
- *Dynamic Degree.* We following existing advances [3, 32, 35] to include Dynamic Degree, which assesses temporal richness and sustained motion in generated videos. This metric complements consistency-based measures, which can otherwise favor overly static generations.

6.2 End-to-End Performance

CineFlow achieves consistent efficiency improvements. Figure 13 shows that CineFlow consistently achieves 1.7–5.5× end-to-end latency speedups compared to existing serving systems. These gains become increasingly pronounced as the target video length scales. While monolithic execution



(a) End-to-end latency on Wan2.2. (b) End-to-end latency on CogVideoX. (c) End-to-end latency on HunyuanVideo.

Figure 13: ***CineFlow consistently achieves the lowest generation latency and enables longer video generation.***

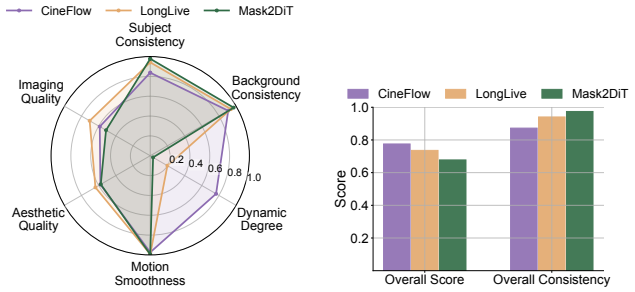
models suffer from the quadratic growth of attention complexity—often leading to severe memory exhaustion (OOM) that caps their scaling capabilities—CineFlow effectively bypasses these bottlenecks. Furthermore, our system minimizes communication overhead by employing online dependency pruning. This removes the necessity for expensive, full-step all-to-all communication that typically dominates the execution time in baseline systems.

CineFlow preserves generation quality while improving motion dynamics. As shown by the VBench results in Figure 14a, CineFlow maintains competitive quality and consistency scores compared to state-of-the-art modeling methods such as LongLive and Mask2DiT, while achieving substantially higher Dynamic Degree. This suggests that CineFlow trades only a nearly negligible drop in consistency for a significant gain in temporal richness and sustained motion, avoiding the overly static generations often favored by consistency-centric baselines. For Quality metrics, LongLive builds on Wan2.1 with its own fine-tuned checkpoints, whereas CineFlow operates on the raw Wan backbone. Therefore, differences in image-quality-related scores are largely affected by fine-tuning and with more data, which is complementary to CineFlow's core contribution. And still, CineFlow achieves 5.4–17.3% better overall score. Crucially, while these baselines require extensive retraining or fine-tuning on large-scale datasets, CineFlow employs a training-free strategy, offering a plug-and-play solution.

Furthermore, CineFlow exhibits a high degree of dynamic adaptability, by respecting the inherent semantic dependency within a prompt. In contrast, traditional monolithic execution models struggle to adapt to multi-scene transitions, often leading to visual artifacts or semantic drift.

A qualitative visualization of these improvements, including generated videos, is provided in the Appendix C.

CineFlow effectively manages scene transitions. We next track video consistency across diverse scene transitions and correlate it with prompt-to-video matching scores. As shown in Figure 15, when prompts exhibit high semantic similarity, CineFlow maintains high subject and background similarity by exploiting temporal dependencies. Conversely, for dissimilar prompt pairs, CineFlow intelligently decouples



(a) VBench dimensions. (b) Quality and consistency.

Figure 14: ***CineFlow maintains quality and consistency while achieving substantially higher Dynamic Degree.***

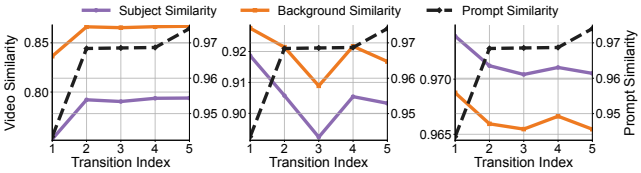
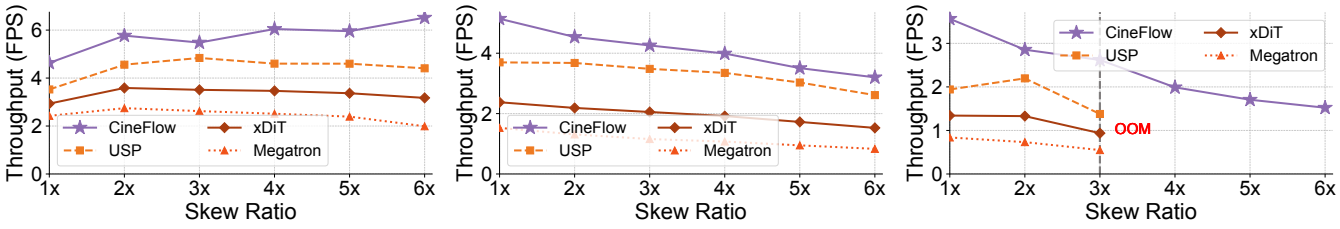


Figure 15: ***Comparison among Mask2DiT, LongLive, and CineFlow on prompt-matching similarity and video consistency under the pairwise transition protocol.***

the generation process to prevent semantic bleed, thereby respecting the transition boundaries. This behavior demonstrates that CineFlow does not merely generate clips in isolation but adaptively applies its fusing and pruning mechanisms to align with the intended narrative flow.

CineFlow achieves better performance under different scene length skew. In practical deployments, scenes within a video often exhibit heterogeneous lengths, leading to significant workload skewness across tasks. To evaluate CineFlow's robustness, we introduce varying skew ratios to the timestamps of each node in the DAG.

As shown in Figure 16, CineFlow consistently delivers throughput gains ranging from 1.15× to 3.27× compared to baselines. Notably, even with a 6× skew ratio, CineFlow maintains high performance whereas baselines suffer from significant throughput degradation and eventual OOM errors. This resilience stems from the scheduler's ability to



(a) Throughput on Wan2.2. (b) Throughput on CogVideoX. (c) Throughput on HunyuanVideo.

Figure 16: *CineFlow maintains high throughput and avoids memory explosion under varying skew ratios.*

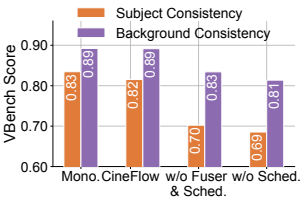


Figure 17: *CineFlow leverages the synergy between the fuser and scheduler to preserve visual coherence.*

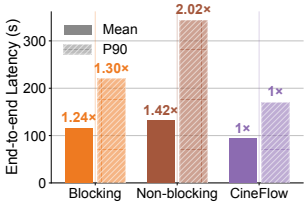


Figure 18: *CineFlow balances end-to-end latency through adaptive synchronization.*

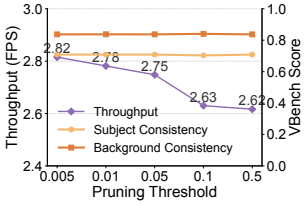


Figure 19: *CineFlow balances throughput and video fidelity.*

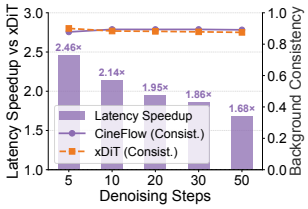


Figure 20: *CineFlow maintains efficiency robustness across denoising steps.*

selectively switch between blocking and non-blocking communication modes and effectively backfill slack time caused by imbalanced task lengths.

6.3 Preformance Breakdown

Quality breakdown. Figure 17 illustrates the individual contribution of each CineFlow component to the overall generation quality. Despite partitioning the monolithic generation into sub-tasks, CineFlow effectively preserves both subject and background consistency, a result primarily driven by the synergy between the fuser and the scheduler. Our result reveals that removing either component leads to a 6.7–14.6% degradation in visual coherence. Notably, disabling the scheduler alone yields even worse performance than disabling both components. This suggests that the diffusion process is highly sensitive to the timing and alignment of fusion; without CineFlow's scheduler to manage blocking and non-blocking timesteps, mismatched fusion interferes with the denoising process. This proves that CineFlow's strategic orchestration, rather than simple merging, is the key to seamless consistency.

Communication breakdown. To evaluate the impact of our scheduling strategy on system performance, we further compare CineFlow against two static baselines: a pure blocking synchronization and a pure non-blocking synchronization (Theorem 4.2). As shown in Figure 18, CineFlow achieves 1.24–1.42× speedup in mean latency over the respective baselines. More importantly, CineFlow significantly mitigates tail latency, reducing P90 latency by 1.30–2.02×. This result provides a key insight: while pure blocking suffers from idle synchronization waits and pure non-blocking leads to severe long-tail slowdown, CineFlow's ability to selectively apply blocking/non-blocking modes allows it to hide communication overhead without sacrificing stability.

Impact of Online Dependency Pruning. Figure 19 illustrates CineFlow's ability to identify and prune redundant dependencies without sacrificing video fidelity. By monitoring latent trajectory similarity and terminating cross-task communication once a pre-defined threshold is exceeded (§4.2), CineFlow maintains consistent subject and background consistency scores. Notably, this adaptive pruning mechanism allows the system to bypass unnecessary synchronization, yielding a nearly 7% throughput speedup. These results demonstrate that CineFlow effectively balances system efficiency and generation quality, ensuring that communication overhead is only incurred when it is semantically essential.

6.4 Sensitivity Analysis and Ablation Studies

Impact of Total Denoising Steps. We next evaluate CineFlow's sensitivity by varying the number of denoising steps. As illustrated in Figure 20, CineFlow consistently outperforms the xDiT baseline across all configurations. Specifically, CineFlow achieves 2.46× speedup at 5 steps and 1.68× at 50 steps. While the speedup ratio slightly decreases as the computational density of the denoising process increases with more steps, CineFlow maintains a high efficiency gain in the most commonly used regimes (e.g., 20–50 steps). Crucially, CineFlow preserves near-perfect background consistency

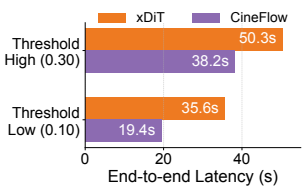


Figure 21: *CineFlow reduces end-to-end latency across compiler thresholds.*

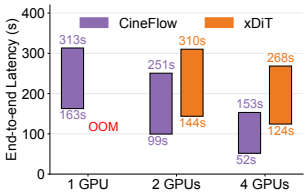


Figure 22: *CineFlow improves scalability and memory efficiency.*

across step counts. This demonstrates that our trajectory-aware fusion successfully decouples generation quality from the specific discretization of the denoising timeline.

Impact of Compiler's Threshold. We evaluate how the Semantic-Dependency Compiler's threshold τ_{merge} (§4.1) influences end-to-end execution time by altering the density of the Semantic DAG. Under a low threshold ($\tau_{merge} = 0.10$), the compiler collapses tasks into fewer, larger nodes, yet CineFlow still achieves a 1.84× speedup over xDiT. This demonstrates that our efficiency gains remain significant even in scenarios with minimal scene-level parallelism.Conversely, as the threshold increases and introduces a more complex, densely connected DAG, CineFlow maintains a 1.32× speedup, proving that our non-blocking execution successfully hide the increased all-to-all communication overhead.

Impact of Cluster Size. Figure 22 shows that CineFlow achieves a consistent 1.55–1.74× speedup compared to xDiT across GPU counts. This superior performance stems from CineFlow's ability to partition complex semantic workflows into manageable subtasks. Notably, while xDiT suffers from OOM errors when restricted to a single GPU for long-video tasks, CineFlow executes successfully by leveraging its dependency-aware task decomposition. Even as the hardware scales to 4 GPUs, CineFlow maintains a significant lead. These results demonstrate CineFlow's capability to maximize limited hardware resources while scaling effectively by overlapping computation with communication.

7 Related Work

Systems for Scaling Video Generation. As sequence lengths grow with video duration, the bidirectional attention and the massive size of spatio-temporal latents frequently hit the memory wall. SageAttention [38, 39] provides 8-bit quantization, while SparseVideoGen [30] exploits spatial-temporal redundancy to skip redundant computations. To scale generation across GPUs, Unified Sequence Parallelism [8], xDiT [7], and DistriFusion [17] split the generation across tokens or patches. However, these approaches operate at the level of *tokens* or *operators*, and still treat video generation as a monolithic process. Instead, CineFlow introduces a *semantic-aware parallelization paradigm* to lift parallelism from low-level tensor partitioning to *dependency-aware task decomposition.*

Long Video Generation. Generating temporally consistent long-form videos remains fundamentally challenging due to the accumulation of structural drift and semantic degradation over extended sampling trajectories. StreamingT2V [11] introduces randomized blending for streaming generation, while Stable Video Infinity [18] employs error recycling to maintain long-range coherence. Other approaches, such as VideoGen-of-Thought [42] and StoryDiffusion [45], incorporate structured reasoning or consistency-aware attention mechanisms to preserve character identity and narrative continuity. However, these approaches focus on improving model behavior. CineFlow takes a complementary systems perspective, unlocking new parallelism while preserving consistency.

Diffusion Model Optimizations. Diffusion models have evolved from convolutional U-Net architectures to DiTs [22] for improved scalability and performance across modalities. Recent models such as Wan [28] and HunyuanVideo [14] adopt transformer backbones and Flow Matching frameworks [19] to process spatiotemporal latents in a unified representation, relying on full bidirectional self-attention to capture global dependencies. Such large-scale transformer structures inherit the high memory and compute demands seen in large language models, necessitating advanced memory management [15] and efficient scheduling of denoising iterations [36]. These challenges motivate system-level innovations that move beyond static operator-level parallelization [1] to mitigate scaling bottlenecks without compromising model quality.

8 Conclusion

This paper presents CineFlow, a dependency-driven inference system for video diffusion. CineFlow reframes the unit of execution from a monolithic spatiotemporal rollout to a semantic dependency graph, exposing new parallelism while preserving global consistency. Its conservative-to-adaptive design combines semantic-dependency compilation, trajectory-aware latent synchronization, and critical-path-aware scheduling to enforce dependencies only when they are needed. Our evaluations show that CineFlow delivers 1.7–5.5× speedup without compromising generation quality.

References

[1] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. 2022. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[2] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural

language inference. In *Conference on Empirical Methods in Natural Language Processing*. https://api.semanticscholar.org/CorpusID:14604520

[3] Jiazi Bu, Pengyang Ling, Pan Zhang, Tong Wu, Xiaoyi Dong, Yuhang Zang, Yuhang Cao, Dahua Lin, and Jiaqi Wang. 2025. Bytheway: Boost your text-to-video generation model to higher quality in a training-free way. In *Proceedings of the Computer Vision and Pattern Recognition Conference*. 12999–13008.

[4] Sitan Chen, Sinho Chewi, Jerry Li, Yuanzhi Li, Adil Salim, and Anru R Zhang. 2022. Sampling is as easy as learning the score: theory for diffusion models with minimal data assumptions. *arXiv preprint arXiv:2209.11215* (2022).

[5] Tsai-Shien Chen, Aliaksandr Siarohin, Willi Menapace, Ekaterina Deyneka, Hsiang-wei Chao, Byung Eun Jeon, Yuwei Fang, Hsin-Ying Lee, Jian Ren, Ming-Hsuan Yang, et al. 2024. Panda-70m: Captioning 70m videos with multiple cross-modality teachers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13320–13331.

[6] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130* (2024).

[7] Jiarui Fang, Jinzhe Pan, Xibo Sun, Aoyu Li, and Jiannan Wang. 2024. xDiT: an Inference Engine for Diffusion Transformers (DiTs) with Massive Parallelism. *arXiv preprint arXiv:2411.01738* (2024).

[8] Jiarui Fang and Shangchun Zhao. 2024. A Unified Sequence Parallelism Approach for Long Context Generative AI. *arXiv preprint arXiv:2405.07719* (2024).

[9] Yu Gao, Haoyuan Guo, Tuyen Hoang, Weilin Huang, Lu Jiang, Fangyuan Kong, Huixia Li, Jiashi Li, Liang Li, Xiaojie Li, et al. 2025. Seedance 1.0: Exploring the boundaries of video generation models. *arXiv preprint arXiv:2506.09113* (2025).

[10] Robert Grandl, Mosharaf Chowdhury, Aditya Akella, and Ganesh Ananthanarayanan. 2016. Altruistic Scheduling in Multi-Resource Clusters. In *OSDI*.

[11] Roberto Henschel, Levon Khachatryan, Hayk Poghosyan, Daniil Hayrapetyan, Vahram Tadevosyan, Zhangyang Wang, Shant Navasardyan, and Humphrey Shi. 2025. StreamingT2V: Consistent, Dynamic, and Extendable Long Video Generation from Text. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2568–2577.

[12] 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. 2023. VBench: Comprehensive Benchmark Suite for Video Generative Models. arXiv:2311.17982 [cs.CV] https://arxiv.org/abs/2311.17982

[13] Akio Kodaira, Chenfeng Xu, Toshiki Hazama, Takanori Yoshimoto, Kohei Ohno, Shogo Mitsuhori, Soichi Sugano, Hanying Cho, Zhijian Liu, and Kurt Keutzer. 2023. StreamDiffusion: A Pipeline-level Solution for Real-time Interactive Generation. (2023). arXiv:2312.12491 [cs.CV]

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

[15] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. (2023).

[16] Holden Lee, Jianfeng Lu, and Yixin Tan. 2022. Convergence for score-based generative modeling with polynomial complexity. *Advances in Neural Information Processing Systems* 35 (2022), 22870–22882.

[17] Muyang Li, Tianle Cai, Jiaxin Cao, Qinsheng Zhang, Han Cai, Junjie Bai, Yangqing Jia, Ming-Yu Liu, Kai Li, and Song Han. 2024. DistriFusion: Distributed Parallel Inference for High-Resolution Diffusion Models. In *CVPR*.

[18] Wuyang Li, Wentao Pan, Po-Chien Luan, Yang Gao, and Alexandre Alahi. 2026. Stable Video Infinity: Infinite-Length Video Generation with Error Recycling. In *The Fourteenth International Conference on Learning Representations*. https://openreview.net/forum?id=X96Ei9n34a

[19] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. 2023. Flow Matching for Generative Modeling. arXiv:2210.02747 [cs.LG] https://arxiv.org/abs/2210.02747

[20] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692 [cs.CL] https://arxiv.org/abs/1907.11692

[21] Yihao Meng, Hao Ouyang, Yue Yu, Qiuyu Wang, Wen Wang, Ka Leong Cheng, Hanlin Wang, Yixuan Li, Cheng Chen, Yanhong Zeng, Yujun Shen, and Huamin Qu. 2025. HoloCine: Holistic Generation of Cinematic Multi-Shot Long Video Narratives. arXiv:2510.20822 [cs.CV] https://arxiv.org/abs/2510.20822

[22] William S. Peebles and Saining Xie. 2022. Scalable Diffusion Models with Transformers. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)* (2022), 4172–4182. https://api.semanticscholar.org/CorpusID:254854389

[23] Adam Polyak, Amit Zohar, Andrew Brown, Andros Tjandra, Animesh Sinha, Ann Lee, Apoorv Vyas, Bowen Shi, Chih-Yao Ma, Ching-Yao Chuang, et al. 2024. Movie Gen: A Cast of Media Foundation Models. *arXiv preprint arXiv:2410.13720* (2024).

[24] Tianhao Qi, Jianlong Yuan, Wanquan Feng, Shancheng Fang, Jiawei Liu, SiYu Zhou, Qian He, Hongtao Xie, and Yongdong Zhang. 2025. Maskˆ 2DiT: Dual Mask-based Diffusion Transformer for Multi-Scene Long Video Generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*. 18837–18846.

[25] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv:1908.10084 [cs.CL] https://arxiv.org/abs/1908.10084

[26] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *ArXiv* abs/1909.08053 (2019). https://api.semanticscholar.org/CorpusID:202660670

[27] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. 2023. Consistency models. (2023).

[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. 2025. Wan: Open and Advanced Large-Scale Video Generative Models. *arXiv preprint arXiv:2503.20314* (2025).

[29] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers. arXiv:2002.10957 [cs.CL] https://arxiv.org/abs/2002.10957

[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. 2025. Sparse Video-Gen: Accelerating Video Diffusion Transformers with Spatial-Temporal

Sparsity. In *Forty-second International Conference on Machine Learning*. https://openreview.net/forum?id=u8CA3qIS0V

[31] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient Streaming Language Models with Attention Sinks. arXiv:2309.17453 [cs.CL] https://arxiv.org/abs/2309.17453

[32] Xin Yan, Yuxuan Cai, Qiuyue Wang, Yuan Zhou, Wenhao Huang, and Huan Yang. 2025. Long video diffusion generation with segmented cross-attention and content-rich video data curation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*. 3184–3194.

[33] Shuai Yang, Wei Huang, Ruihang Chu, Yicheng Xiao, Yuyang Zhao, Xianbang Wang, Muyang Li, Enze Xie, Yingcong Chen, Yao Lu, et al. 2025. Longlive: Real-time interactive long video generation. *arXiv preprint arXiv:2509.22622* (2025).

[34] Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. 2024. CogVideoX: Text-to-Video Diffusion Models with An Expert Transformer. *arXiv preprint arXiv:2408.06072* (2024).

[35] Aoxiong Yin, Xu Tan, Kai Shen, Yichong Leng, Xinyu Zhou, Juncheng Li, and Siliang Tang. 2025. The best of both worlds: Integrating language models and diffusion models for video generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 15604–15615.

[36] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 521–538. https://www.usenix.org/conference/osdi22/presentation/yu

[37] Zhou Yu, Dejing Xu, Jun Yu, Ting Yu, Zhou Zhao, Yueting Zhuang, and Dacheng Tao. 2019. Activitynet-qa: A dataset for understanding complex web videos via question answering. In *Proceedings of the AAAI conference on artificial intelligence*. 9127–9134.

[38] Jintao Zhang, Haofeng Huang, Pengle Zhang, Jia wei, Jun Zhu, and Jianfei Chen. 2025. SageAttention2: Efficient Attention with Thorough Outlier Smoothing and Per-thread INT4 Quantization. In *Forty-second International Conference on Machine Learning*. https://openreview.net/forum?id=nC8XliUxeg

[39] Jintao Zhang, Jia wei, Pengle Zhang, Jun Zhu, and Jianfei Chen. 2025. SageAttention: Accurate 8-Bit Attention for Plug-and-play Inference Acceleration. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=OL44KtasKc

[40] Peixuan Zhang, Zijian Jia, Kaiqi Liu, Shuchen Weng, Si Li, and Boxin Shi. 2025. STAGE: Storyboard-Anchored Generation for Cinematic Multi-shot Narrative. (2025).

[41] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. (2023).

[42] Mingzhe Zheng, Yongqi Xu, Haojian Huang, Xuran Ma, Yexin Liu, Wenjie Shu, Yatian Pang, Feilong Tang, Qifeng Chen, Harry Yang, et al. 2024. VideoGen-of-Thought: Step-by-step generating multi-shot video with minimal manual intervention. *arXiv preprint arXiv:2412.02259* (2024).

[43] Zangwei Zheng, Xiangyu Peng, Yuxuan Lou, 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. 2026. Open-Sora 2.0: Training a Commercial-Level Video Generation Model in $200k. arXiv:2503.09642 [cs.GR] https://arxiv.org/abs/2503.09642

[44] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*.

[45] Yupeng Zhou, Daquan Zhou, Ming-Ming Cheng, Jiashi Feng, and Qibin Hou. 2024. Storydiffusion: Consistent self-attention for long-range image and video generation. *Advances in Neural Information Processing Systems* 37 (2024), 110315–110340.

A Appendix: Bounded Quality Gap Between CineFlow and Monolithic Long-Video Generation

This appendix studies a comparative question: how far can the executed CineFlow trajectory deviate from the monolithic long-video execution of the *same* pretrained generator? Our goal is not to bound either method against the unknown true data-generating process. Instead, we show that CineFlow is a controlled approximation to monolithic execution: its quality gap to the monolithic baseline is bounded by two design-dependent quantities: structural approximation and fusion approximation.

A.1 Comparison Scope and Dynamics

Let

$$x^M(t) = \big(x_1^M(t), x_2^M(t), \dots, x_n^M(t)\big)$$

denote the joint latent state produced by monolithic long-video generation, where $x_i^M(t)$ is the latent for shot i at denoising time $t \in [0, T]$.

Monolithic execution. We model direct long-video generation as the coupled latent dynamics

$$\frac{dx_i^M(t)}{dt} \;=\; f_i\big(x_i^M(t), t; c_i\big) + \sum_{j \neq i} G_{ji}\big(x_j^M(t), x_i^M(t), t\big). \quad (1)$$

Here f_i is the local drift for node i, while G_{ji} denotes the cross-shot coupling from node j to node i induced by full-sequence generation.

Executed CineFlow. Let

$$\hat{x}(t) = \big(\hat{x}_1(t), \hat{x}_2(t), \dots, \hat{x}_n(t)\big)$$

denote the latent trajectory produced by CineFlow. For each node i, let $\mathrm{Pa}(i)$ be its semantic parent set, and let $A_i(t) \subseteq \mathrm{Pa}(i)$ be the subset of parent edges that are active at time t. The executed dynamics are

$$\frac{d\hat{x}_i(t)}{dt} \;=\; f_i\big(\hat{x}_i(t), t; c_i\big) + \sum_{j \in A_i(t)} \bar{w}_{ji}(t)\, \widetilde{G}_{ji}\big(\hat{x}_j(t), \hat{x}_i(t), t\big), \quad (2)$$

where $\bar{w}_{ji}(t)$ are normalized active-edge weights. If $A_i(t) = \emptyset$, the coupling term in Eq. (2) is defined as zero.

Joint vector fields. Let $F^M(x,t)$ and $F^C(x,t)$ denote the joint vector fields induced by Eqs. (1)–(2), respectively. The appendix compares the two trajectories $x^M(t)$ and $\hat{x}(t)$ generated from the same prompt and the same initial latent.

A.2 Assumptions

Assumption A1 (common initialization and well-posedness). The two executions start from the same latent state:

$$x^M(0) = \hat{x}(0) = x_0.$$

Moreover, the induced dynamics remain well posed on the reachable set over $[0, T]$.

Assumption A2 (Lipschitz stability of executed dynamics). There exists $L > 0$ such that for all reachable states x, y and all $t \in [0, T]$,

$$\big\|F^C(x,t) - F^C(y,t)\big\|_\infty \;\le\; L\,\|x - y\|_\infty. \quad (A2)$$

This assumption states that the executed CineFlow dynamics do not amplify a small latent perturbation arbitrarily fast. Lipschitz-type stability of the denoising dynamics is the standard assumption in convergence analyses of diffusion sampling [4, 16].

Assumption A3 (quality continuity). Let Q be a terminal quality functional used to compare outputs. We assume that Q is Lipschitz with respect to the terminal latent under the $\|\cdot\|_\infty$ norm:

$$\big|Q(x) - Q(y)\big| \;\le\; K_Q\,\|x - y\|_\infty. \quad (A3)$$

This keeps the theorem agnostic to the particular choice of evaluation metric: the argument only needs continuity of the terminal quality readout with respect to the latent.

A.3 Exact Decomposition of the Drift Gap

Node-wise drift gap. For node i, define the monolithic-to-CineFlow drift mismatch at the same state x and time t by

$$\Delta_i(x,t) \;:=\; \sum_{j \neq i} G_{ji}(x_j, x_i, t) \;-\; \sum_{j \in A_i(t)} \bar{w}_{ji}(t)\, \widetilde{G}_{ji}(x_j, x_i, t). \quad (3)$$

This quantity measures how much coupling CineFlow omits or approximates at node i relative to monolithic execution.

Lemma A.1 (Exact drift-gap decomposition)**.** *For every node i, the drift gap admits the decomposition*

$$\Delta_i = \underbrace{\sum_{j \neq i} G_{ji} - \sum_{j \in A_i(t)} G_{ji}}_{\Delta_i^{\mathrm{struct}}} + \underbrace{\sum_{j \in A_i(t)} \Big(G_{ji} - \bar{w}_{ji}(t)\widetilde{G}_{ji}\Big)}_{\Delta_i^{\mathrm{fuse}}}. \quad (4)$$

Proof. Split the monolithic coupling into nodes outside and inside the semantic parent set:

$$\sum_{j \neq i} G_{ji} = \sum_{j \notin \mathrm{Pa}(i)} G_{ji} + \sum_{j \in \mathrm{Pa}(i)} G_{ji}.$$

Then partition the parent-set sum into active and inactive parents:

$$\sum_{j \in \mathrm{Pa}(i)} G_{ji} = \sum_{j \in \mathrm{Pa}(i) \setminus A_i(t)} G_{ji} + \sum_{j \in A_i(t)} G_{ji}.$$

Substituting into Eq. (3) gives

$$\Delta_i = \sum_{j \notin \mathrm{Pa}(i)} G_{ji} + \sum_{j \in \mathrm{Pa}(i) \setminus A_i(t)} G_{ji} + \sum_{j \in A_i(t)} G_{ji} - \sum_{j \in A_i(t)} \bar{w}_{ji}(t)\widetilde{G}_{ji},$$

which is exactly Eq. (4). □

Remark (fusion error has two sources). The fusion term can be further split as

$$\Delta_i^{\text{fuse}} = \underbrace{\sum_{j\in A_i(t)} \big(1-\bar{w}_{ji}(t)\big)G_{ji}}_{\Delta_i^{\text{wt}}} + \underbrace{\sum_{j\in A_i(t)} \bar{w}_{ji}(t)\Big(G_{ji}-\widetilde{G}_{ji}\Big)}_{\Delta_i^{\text{op}}}. \tag{5}$$

The first part captures distortion introduced by weight normalization on the active parent set. The second captures operator mismatch between the exact coupling G_{ji} and the fuser-induced approximation $\widetilde{G}_{ji}$. Both are absorbed into the aggregate fusion error below.

A.4 Pointwise Structural Bounds

Assumption B1 (structural approximation error). For each node i, all reachable states x, and all $t \in [0,T]$,

$$\left\| \sum_{j\neq i} G_{ji}(x_j,x_i,t) - \sum_{j\in A_i(t)} G_{ji}(x_j,x_i,t) \right\| \le \varepsilon_{\text{struct},i}(t). \tag{B1}$$

This term captures the coupling omitted by semantic decomposition and adaptive dependency pruning.

Assumption B2 (fusion-approximation error). For each node i, all reachable states x, and all $t \in [0,T]$,

$$\left\| \sum_{j\in A_i(t)} \Big(G_{ji}(x_j,x_i,t) - \bar{w}_{ji}(t)\widetilde{G}_{ji}(x_j,x_i,t)\Big) \right\| \le \varepsilon_{\text{fuse},i}(t). \tag{B2}$$

This term aggregates both the weight-distortion error and the operator approximation error in Eq. (5).

Proposition A.2 (System-level pointwise structural gap)**.** *For any reachable state x and any $t \in [0,T]$, define*

$$\delta(x,t) := \big\|F^M(x,t) - F^C(x,t)\big\|_\infty.$$

Under Assumptions (B1)–(B2),

$$\delta(x,t) \le \bar{\varepsilon}_{\text{struct}}(t) + \bar{\varepsilon}_{\text{fuse}}(t), \tag{6}$$

$$\bar{\varepsilon}_{\text{struct}}(t) := \max_i \varepsilon_{\text{struct},i}(t),$$
$$\bar{\varepsilon}_{\text{fuse}}(t) := \max_i \varepsilon_{\text{fuse},i}(t).$$

Proof. For each node i, Lemma A.1 and the triangle inequality give

$$\|\Delta_i(x,t)\| \le \|\Delta_i^{\text{struct}}(x,t)\| + \|\Delta_i^{\text{fuse}}(x,t)\|.$$

Applying Assumptions (B1)–(B2),

$$\|\Delta_i(x,t)\| \le \varepsilon_{\text{struct},i}(t) + \varepsilon_{\text{fuse},i}(t).$$

Taking the maximum over i yields Eq. (6). □

A.5 Bounded Quality Gap to Monolithic Generation

Theorem A.3 (Bounded quality gap to monolithic generation)**.** *Let $x^M(t)$ be the monolithic latent trajectory defined by Eq. (1), and let $\hat{x}(t)$ be the CineFlow trajectory defined by Eq. (2). Suppose Assumptions A1–A3 and Assumptions B1–B2 hold. Define the terminal quality gap*

$$\Delta_Q(T) := \big|Q(\hat{x}(T)) - Q(x^M(T))\big|.$$

Then

$$\Delta_Q(T) \le K_Q \int_0^T e^{L(T-s)}\Big(\bar{\varepsilon}_{\text{struct}}(s) + \bar{\varepsilon}_{\text{fuse}}(s)\Big)\,ds. \tag{7}$$

In particular, if the two terms admit uniform bounds

$$\bar{\varepsilon}_{\text{struct}}(s) \le \bar{\varepsilon}_{\text{struct}}, \qquad \bar{\varepsilon}_{\text{fuse}}(s) \le \bar{\varepsilon}_{\text{fuse}},$$

then

$$\big|Q(\hat{x}(T)) - Q(x^M(T))\big| \le K_Q \frac{e^{LT}-1}{L}\Big(\bar{\varepsilon}_{\text{struct}} + \bar{\varepsilon}_{\text{fuse}}\Big). \tag{8}$$

Proof. Define the trajectory error

$$e(t) := x^M(t) - \hat{x}(t).$$

Then

$$\dot{e}(t) = F^M(x^M(t),t) - F^C(\hat{x}(t),t).$$

Add and subtract $F^C(x^M(t),t)$ to obtain

$$\dot{e}(t) = \big(F^M(x^M(t),t) - F^C(x^M(t),t)\big) + \big(F^C(x^M(t),t) - F^C(\hat{x}(t),t)\big).$$

Taking the upper Dini derivative and using Assumption A2,

$$D^+\|e(t)\|_\infty \le \delta(x^M(t),t) + L\,\|e(t)\|_\infty.$$

Because $e(0)=0$ by Assumption A1, Grönwall's inequality yields the standard perturbation bound for ODE trajectories (cf. the analogous argument for probability-flow ODE trajectories in [27]):

$$\|e(T)\|_\infty \le \int_0^T e^{L(T-s)}\,\delta(x^M(s),s)\,ds.$$

Applying Proposition A.2,

$$\|e(T)\|_\infty \le \int_0^T e^{L(T-s)}\cdot\Big(\bar{\varepsilon}_{\text{struct}}(s) + \bar{\varepsilon}_{\text{fuse}}(s)\Big)\,ds.$$

Finally, Assumption A3 gives

$$\big|Q(\hat{x}(T)) - Q(x^M(T))\big| \le K_Q\,\|e(T)\|_\infty,$$

which proves Eq. (7). Equation (8) follows by replacing the time-varying terms by their uniform upper bounds and evaluating the integral. □

Interpretation. Theorem A.3 states that CineFlow is a bounded perturbation of monolithic long-video generation. Its deviation is controlled by two design-dependent quantities:

- $\bar{\varepsilon}_{\text{struct}}$: approximation introduced by semantic decomposition and adaptive pruning;
- $\bar{\varepsilon}_{\text{fuse}}$: approximation introduced by weighted fusion on active edges.

The theorem does not claim that CineFlow must match monolithic execution exactly. It claims something narrower and more useful: the quality gap to the monolithic baseline cannot grow arbitrarily as long as these two quantities remain controlled.

A.6 Heuristic Dependence on the Number of Chunks

The theorem above gives a clean comparative upper bound, but it does not by itself prescribe the optimal number of chunks. To reason about chunk count, we introduce the shorthand

$$B_n(T) \;:=\; K_Q \,\frac{e^{LT}-1}{L}\Big(\eta_{\text{struct}}(n)+\eta_{\text{fuse}}(n)\Big), \tag{9}$$

where $\eta_{\text{struct}}(n)$ and $\eta_{\text{fuse}}(n)$ are empirical or model-based summaries of the two error sources as functions of the chunk count n.

Qualitative dependence on n. Increasing n usually creates more chunk boundaries and thus more cross-boundary communication events. Consequently, both the communication control term and the fusion term often increase under finer chunking. The sparsification term depends on compiler granularity. A finer decomposition may expose more opportunities to drop inter-node coupling, whereas a coarser decomposition may reduce parallelism while preserving more coupling within each chunk. The trade-off among these effects is a design issue rather than a theorem.

Operational interpretation of the two terms. In practice, the two summaries in Eq. (9) are driven by concrete knobs:

- The compiler and the pruning policy determine $\eta_{\text{struct}}(n)$.
- The fusion operator and coupling strength determine $\eta_{\text{fuse}}(n)$.

This is why the theorem is most useful as a design guide: it identifies which parts of the system must stay small in order to keep the gap to monolithic generation small.

A.7 Practical Advantage Beyond the Comparative Bound

Remark (possible practical advantage). The comparative theorem above only upper bounds the deviation of CineFlow from monolithic execution. It does *not* say that CineFlow must be worse in practice. A shorter chunk length may operate closer to the model's native training-length regime, while monolithic long-video execution may run further out of distribution. That effect is empirical and model dependent, so we keep it separate from the formal theorem.

Limitations of the appendix bound. The appendix analyzes the gap between two execution policies of the same pretrained generator under a fixed semantic DAG and the same initial latent. It does not prove compiler optimality, does not bound the generator's distance to the true data distribution, and does not derive a theorem for the optimal chunk count. Those questions are important, but they are distinct from the comparative bounded-gap statement established here.

B Appendix: Efficiency Bound Relative to Monolithic Execution

This appendix studies the efficiency side of the same comparison. We do not attempt to prove that CineFlow always dominates monolithic generation on every hardware stack. Instead, we show a narrower systems statement: under the design described in §2 and §4, CineFlow avoids the quadratic latency growth of monolithic long-video execution by keeping each subtask length bounded and by limiting cross-subtask communication to a sparse, temporally truncated dependency graph.

B.1 Setup

Fix a spatial resolution $H \times W$ and a denoising step count S. Let F_{tot} denote the total number of frames in the target long video.

Monolithic execution. The monolithic baseline generates the entire video as a single DiT sequence of length proportional to F_{tot}. As discussed in §2.1, full bidirectional attention is computed over all video tokens at every denoising step.

Executed CineFlow. Suppose CineFlow decomposes the request into K non-empty subtasks with frame counts

$$f_1,\dots,f_K \in \mathbb{N}_{>0}, \qquad \sum_{i=1}^{K} f_i = F_{\text{tot}}, \qquad f_i \le m.$$

where m is the maximum clip length selected by the compiler and scheduler. Following §4.3, we assume execution on N GPUs, so at most N subtasks can make forward progress concurrently in one scheduling round.

Let M denote the number of scheduling rounds taken by the executed CineFlow schedule on the realized semantic DAG. We only use two generic facts about M: first, M depends on both the dependency structure and the scheduler; second, since every non-empty round completes at least one subtask, we always have

$$1 \le M \le K.$$

Let $L_{\text{clip}}(F)$ denote the latency of generating a single clip of F frames under the fixed (H, W, S) setting, excluding one-time model setup. Let $L_{\text{mono}}(F_{\text{tot}})$ and $L_{CineFlow}(F_{\text{tot}}; m, N)$ denote the end-to-end latencies of monolithic execution and CineFlow, respectively, with all one-time initialization costs absorbed into an additive constant B_0.

Communication model. For each semantic dependency edge (j, i) retained by the compiler, let $r_{ji} \in \{0, 1, \ldots, S\}$ denote the number of denoising steps during which the edge is actually active for cross-subtask conditioning. This captures both the communication schedule and trajectory-aware pruning from §4.2: if an edge is never activated then $r_{ji} = 0$, while an edge that remains active for only the first few denoising steps has $r_{ji} \ll S$.

We write $d_{\max}$ for the maximum indegree of the executed semantic DAG and assume there is a constant $C_{\text{comm}}(m)$ such that one active edge-step communication event between clips of length at most m contributes at most $C_{\text{comm}}(m)$ additional latency. The scheduler's locality-aware placement can only reduce this constant by co-locating dependent tasks; it does not worsen the asymptotic bound.

B.2 The Quadratic Trap of Monolithic Long-Video Generation

Lemma B.1 (Single-clip latency scaling). *For fixed (H, W, S), there exist constants $\alpha_{\text{lo}} > 0$, $\alpha_{\text{hi}} > 0$, $\beta \geq 0$, and $\gamma \geq 0$ such that*

$$S\,\alpha_{\text{lo}} F^2 \;\leq\; L_{\text{clip}}(F) \;\leq\; S\big(\alpha_{\text{hi}} F^2 + \beta F + \gamma\big)$$

for every feasible clip length $F \geq 1$.

Proof. At fixed spatial resolution, the number of video tokens is linear in the frame count: $N(F) = \rho(H, W)\, F$ for some resolution-dependent constant $\rho(H, W) > 0$. Each denoising step applies full bidirectional attention over this token set, so the dominant attention term scales as $\Theta\big(N(F)^2\big) = \Theta(F^2)$. The remaining per-step operations, including projections, normalization, and feed-forward layers, contribute at most linearly in $N(F)$ and therefore at most linearly in F. Multiplying by the fixed denoising horizon S yields the claimed constants. The lower bound holds because attention is the computational bottleneck: even after accounting for linear-cost layers the per-step cost is at least $\alpha_{\text{lo}} F^2$ for a suitable $\alpha_{\text{lo}} > 0$ that absorbs $\rho(H, W)^2$ and the attention constant. □

Theorem B.2 (Monolithic quadratic trap). *Under the setting above,*

$$L_{\text{mono}}(F_{\text{tot}}) \;\geq\; B_0 + S\,\alpha_{\text{lo}} F_{\text{tot}}^2. \tag{10}$$

In particular, monolithic long-video generation has quadratic latency growth in the target frame budget.

Proof. Monolithic execution generates the full F_{tot}-frame sequence in a single call, so Lemma B.1 applies with $F = F_{\text{tot}}$. Adding the one-time setup constant B_0 gives Eq. (10). □

B.3 Bounded-Clip Execution with Sparse Communication

Theorem B.3 (Linear upper bound for CineFlow). *Under the setup above,*

$$\begin{aligned} L_{CineFlow}(F_{\text{tot}}; m, N) \;&\leq B_0 + MS\big(\alpha_{\text{hi}} m^2 + \beta m + \gamma\big) \\ &\quad + C_{\text{comm}}(m) \sum_{(j,i)\in E} r_{ji}, \end{aligned} \tag{11}$$

where E is the executed semantic edge set.

If, in addition,

$$r_{ji} \leq R \qquad \text{for all } (j, i) \in E$$

for some communication horizon $R \leq S$, then

$$\begin{aligned} L_{CineFlow}(F_{\text{tot}}; m, N) \;&\leq B_0 + MS\big(\alpha_{\text{hi}} m^2 + \beta m + \gamma\big) \\ &\quad + C_{\text{comm}}(m)\, R\, d_{\max}\, K. \end{aligned} \tag{12}$$

Since $M \leq K \leq F_{\text{tot}}$ for non-empty subtasks, it follows that

$$L_{CineFlow}(F_{\text{tot}}; m, N) = O(F_{\text{tot}})$$

for fixed m, N, R, and $d_{\max}$.

Proof. Because $f_i \leq m$, Lemma B.1 yields

$$L_{\text{clip}}(f_i) \leq S\big(\alpha_{\text{hi}} m^2 + \beta m + \gamma\big)$$

for every i. By definition, the executed schedule uses M rounds. Within each round, the round latency is determined by the slowest clip scheduled in that round; since every clip satisfies $f_i \leq m$, this worst-case per-round cost is at most $L_{\text{clip}}(m)$. The clip-generation component of the latency is therefore upper bounded by

$$MS\big(\alpha_{\text{hi}} m^2 + \beta m + \gamma\big).$$

This is conservative: it only assumes that each round contains clips of length at most m, not that the scheduler achieves perfect work-conserving packing. The dependence on N is therefore implicit through the realized round count M.

Now consider cross-subtask communication. By construction, an edge (j, i) incurs communication overhead only on the denoising steps when it is active, namely r_{ji} times in total. Since one active edge-step event contributes at most $C_{\text{comm}}(m)$ latency, the total communication overhead is at most

$$C_{\text{comm}}(m) \sum_{(j,i)\in E} r_{ji}.$$

Note that we account for communication cost additively (i.e., serialized with computation). When the scheduler overlaps communication with the next round's computation, the actual latency is lower, so the bound remains valid. Adding the setup constant B_0 proves Eq. (11).

If every executed edge is active for at most R denoising steps, then

$$\sum_{(j,i)\in E} r_{ji} \leq R\,|E|.$$

Moreover, a DAG on K nodes with maximum indegree $d_{\max}$ satisfies $|E| \leq d_{\max} K$. Substituting these inequalities into

Eq. (11) yields Eq. (12). Finally, since $M \le K$ and $K \le F_{\text{tot}}$ for non-empty subtasks, the right-hand side is affine in F_{tot} when m, N, R, and $d_{\max}$ are fixed. □

Interpretation. The efficiency theorem isolates the two mechanisms that prevent CineFlow from falling into the monolithic quadratic trap:

- bounded clip length m, which caps the quadratic attention cost of each subtask; and
- sparse communication, which keeps cross-subtask consistency overhead proportional to the number of active dependency-edge steps rather than to the full long-video token sequence at every denoising step.

This matches the design in the main paper: the compiler controls K, the fuser controls the active edge-step counts r_{ji} through temporal activation and pruning, and the scheduler controls the effective concurrency through the available GPU count N and locality-aware placement, which jointly determine the realized round count M.

Relation to the main-text knobs. The bound exposes the same trade-offs discussed in §4. A finer compiler decomposition increases K; a longer communication window increases the active-step counts r_{ji}, while more aggressive pruning reduces them; and better placement shrinks the constant $C_{\text{comm}}(m)$ by converting remote communication into local access. In the concrete schedule discussed in §4.2, restricting communication to the first few denoising steps corresponds exactly to keeping R small in Eq. (12). Meanwhile, higher concurrency and better overlap show up indirectly by reducing the realized number of rounds M toward its hardware-imposed lower limit.

Corollary B.4 (Asymptotic gap to monolithic latency)**.** *Under the assumptions of Theorems B.2 and B.3, there exist constants $c_0, c_1 > 0$ such that*

$$\frac{L_{\text{mono}}(F_{\text{tot}})}{L_{CineFlow}(F_{\text{tot}}; m, N)} \;\ge\; \frac{c_1 F_{\text{tot}}^2}{c_0 + F_{\text{tot}}},$$

so the speedup ratio grows at least linearly in F_{tot}.

Proof. Set $c_1 = S\,\alpha_{\text{lo}}$. From Theorem B.2, $L_{\text{mono}}(F_{\text{tot}}) \ge B_0 + c_1 F_{\text{tot}}^2 \ge c_1 F_{\text{tot}}^2$. From Eq. (12) together with $M \le K \le F_{\text{tot}}$ for non-empty subtasks, $L_{CineFlow}(F_{\text{tot}}; m, N)$ is bounded above by

$$B_0 + F_{\text{tot}}\, S(\alpha_{\text{hi}} m^2 + \beta m + \gamma) + C_{\text{comm}}(m)\, R\, d_{\max}\, F_{\text{tot}}.$$

Hence there exists a constant $c_0 > 0$ such that

$$L_{CineFlow}(F_{\text{tot}}; m, N) \le c_0\,(1 + F_{\text{tot}}) \qquad \text{for all } F_{\text{tot}} \ge 1.$$

Taking the ratio and absorbing constants proves the claim. □

Remark (what this appendix does and does not claim). This efficiency appendix proves a bounded asymptotic comparison against monolithic execution of the same generator. It does not claim a universal speedup factor for every prompt or every hardware stack, and it does not replace the empirical latency measurements in §6. Its role is narrower: to show that the design in §4 changes the scaling law from quadratic-in-length monolithic execution to linear-in-length bounded-clip execution, provided that the communication horizon and executed DAG sparsity remain controlled. Crucially, the $O(F_{\text{tot}})$ result requires the maximum clip length m to be a *fixed* design parameter independent of F_{tot}. If m were allowed to grow with F_{tot}, the per-clip quadratic cost would re-emerge; keeping m bounded is a deliberate design choice enforced by the compiler and scheduler.

C Appendix: Qualitative Comparison of Motion Dynamics.

CineFlow enables realistic temporal evolution through dependency-aware decomposition. As illustrated in Figure 23, traditional monolithic execution models often struggle to maintain motion intensity over long durations, resulting in "static-video" artifacts where the subject appears frozen (top row). In contrast, CineFlow leverages its semantic-aware parallelization to decouple global consistency from local execution characteristics. By treating the generation as a series of dependency-aware tasks rather than a single rigid sequence, CineFlow successfully captures richer narrative elements—such as the fluctuating sea state and dynamic sailboat orientations—ensuring that the generated video is both temporally stable and visually vibrant. This proves that our orchestration strategy provides the necessary flexibility for models to express complex motion priors that are otherwise suppressed in monolithic paradigms.

D Appendix: Evaluation of Dependency Compiler Reliability

To address concerns regarding the reliability of the Dependency Compiler, we provide additional details on the "reorder accuracy" experiment illustrated in Figure 9b. We curated a dedicated benchmark comprising 100 complex video prompts that feature diverse multi-scene narrative patterns, specifically focusing on cases requiring reordering versus those that do not. Accuracy was measured against ground-truth dependencies manually annotated via GPT-4 to identify "visual materialization" errors—instances where new elements appear without proper prior establishment.

To ensure a rigorous evaluation of the baselines, the general LLMs (Gemini and Qwen) were provided with the specialized Consistency Analysis Prompt detailed below:

```
You are a video-generation consistency analyst.

You will be given:
```

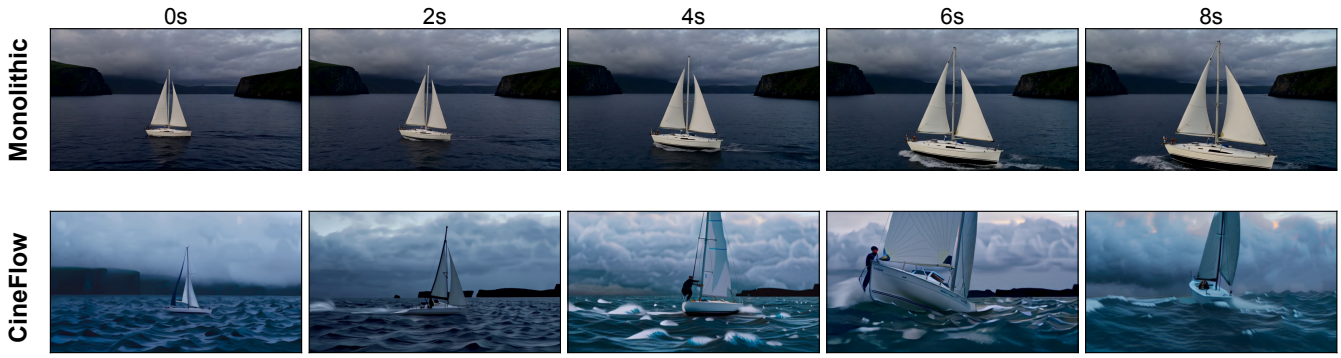


Figure 23: ***CineFlow preserves high motion dynamics and perspective diversity while maintaining long-term consistency. Compared to the Monolithic baseline, which produces static, photo-like segments (e.g., the repetitive boat pose from 0s to 8s), CineFlow adaptively manages scene transitions to depict realistic wave turbulence and varied camera angles without sacrificing subject identity.***

```
  - "original_prompt": the original short video description (ground
      truth of what should appear)
  - "pattern": the narrative structure type (Linear / Convergence /
      Recurrent / Expansion)
  - "scenes": a list of sentences or short segments from the expanded
      description,
    in the ORDER they appear in the script. These correspond to
      sequential visual moments.

Your task:
  Identify whether any scene (index >= 1) introduces a SIGNIFICANT
      NEW VISUAL ELEMENT
  that was NOT established or implied in the original_prompt or any
      earlier scene.

  This matters because if a video is generated strictly in sequence,
      any element that
  appears "out of nowhere" will seem to materialize suddenly - a
      visual consistency error.
  Such cases need reordering (establishing the element earlier)
      before generation.

A "significant new visual element" requiring reorder is:
  - A new character, person, or animal not mentioned in
      original_prompt or prior scenes
  - A new major prop or object that is central to the action
  - A new location or setting component that changes where the scene
      takes place

Do NOT flag:
  - Camera movements, lighting changes, mood/atmosphere descriptions
  - Close-ups or angle changes on already-established elements
  - Details that naturally belong to the established scene (e.g. "her
      hair blows" if a woman was already introduced)
  - Elements directly mentioned in original_prompt (even if first
      described later)

Respond ONLY with a JSON object:
{
  "has_reorder": true | false,
  "new_elements": [
    {
      "scene_idx": <int, 0-based>,
      "element": "<short label of the new element>",
      "reason": "<one sentence: why this element is unexpected given
          prior context>"
    }
  ]
}
```

The observed performance gap between CineFlow (65%) and general-purpose LLMs such as Gemini (63%) and Qwen (52%) stems from the specialized nature of video consistency logic rather than a lack of general intelligence in those models. While general LLMs are proficient in narrative continuity, they often struggle with specific visual establishment requirements; for instance, a character entering a room may be narratively sound but remains visually inconsistent if that character was not pre-buffered in the latent space. Our compiler excels because it utilizes a pairwise entailment formulation via MiniLM, which is explicitly trained to detect fine-grained semantic dependencies. Unlike generative LLMs that may hallucinate temporal flows, our deterministic pre-analysis focuses strictly on anchor-based visual consistency.