dKV Forcing: Separating Generation and Memory Representations through a Differentiable KV Cache

Zhiqi Li Bo Zhu
Georgia Institute of Technology
zli3167@gatech.edu

Project page: https://zhiqili-cg.github.io/dKV-Forcing-Project
Code: https://github.com/ZhiqiLi-CG/dKV-Forcing



Figure 1. **Trained on 5 s clips, dKV Forcing generates up to four minutes.** The top two rows are 60 s generations and the bottom two are 240 s generations, 12 and 46 times the training clip, with the time stamp drawn in each frame. The prompts are a cat and a dog in a sword fight, a child blowing bubbles, a cat drinking water, and a man wiping a kitchen counter. Subject, layout and lighting hold throughout while the motion continues; the same generations are compared with CF++ and SGF in Figures 4 and 6 and Appendix C.5.

Abstract

A causal video generator produces two key–value (KV) states for every segment: the KV it uses while denoising that segment, and the KV it writes into the cache for later segments to read. In a pretrained generator the two already differ substantially, later segments attend to them differently, and replacing the written KV with the generation KV degrades the continuation. We propose dKV Forcing (dKV denotes differentiable KV) to learn a dedicated memory representation from its future use: a writer-specific low-rank residual on the shared backbone separates memory writing from generation and is trained through the future reads of what it writes. Learning a write from its future use needs gradients through the cache, which the mutable cache in common use cannot provide. dKV Forcing replaces the training-time cache by persistent differentiable pages with local rematerialization and, in one temporal traversal, computes exactly the gradient that a two-pass reconstruction of the same objective computes. One traversal computes the full memory gradient at 7% *generator-update overhead over detached-write training, with* 28% *less resident memory and* 20.8% *shorter training cycles than the two-pass reconstruction. Over Self Gradient Forcing (SGF), the same objective trained with the same traversal, writer adaptation raises VBench-Long aesthetic quality by* 6.6 *points at 60 s and* 8.5 *points at 240 s with the highest subject and background consistency, and rollout dropout of the residual keeps most of this gain while restoring the motion of training without write gradients.*

1. Introduction

Video diffusion transformers produce convincing clips, but they denoise every frame of a clip jointly, so generating one more second means recomputing the whole sequence [19]. Causal variants remove this constraint: they generate a video segment by segment (a latent frame, or a short chunk of frames), each new segment attending to a KV cache of

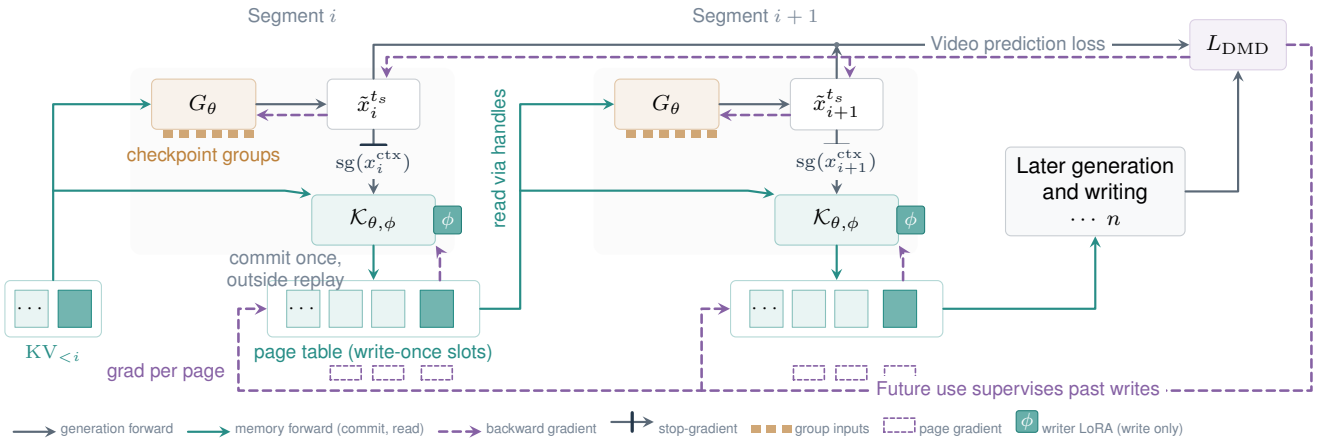


Figure 2. **dKV Forcing: generation, memory writing and the memory gradient in one temporal traversal.** For each segment the model runs a generation forward G_θ, whose prediction $\tilde{x}_i^{t_s}$ enters the video loss, and a memory write $\mathcal{K}_{\theta,\phi}$, which turns the detached prediction $\text{sg}(x_i^{\text{ctx}})$ into the KV of that segment; θ is the shared backbone and ϕ the writer LoRA on the K and V projections, active only at the write. The written KV is committed once to a fresh slot of the page table and never modified afterwards. Later segments read the stored prefix through handles (teal). Both forwards are checkpointed in groups of blocks (gold ticks). In backward (purple, dashed) the gradient of the loss reaches every prediction, flows through the handles to every page it read, and on to the writer that produced that page; the stop-gradient keeps it from re-entering the prediction, and replay only reads the pages. Sampling substeps without gradient and the final, never-read write are omitted. Implementation details of each element are in Appendix B.4.

the segments already produced, and reach streaming speed on a single GPU [1, 7, 26]. In such a model the cache is the memory. What a later segment knows about the past is exactly what earlier segments wrote into the cache.

Training these models on the histories they generate themselves is what made them usable at length: Self Forcing rolls the model out during training and supervises the rollout with a video-level distribution-matching objective [7, 24, 25], removing the train–test gap that otherwise accumulates error over long generations. Building on it, Causal Forcing makes the teacher itself causal [34], and Causal Forcing++ accelerates the initialization through causal consistency distillation [32]. Their gradient, however, stops at the cache. Each segment is trained to generate well given the history it is handed, but no gradient reaches the operations that produced that history, so the model is never taught how to remember. The KV written into the cache comes from the same projections that denoise the current segment, and those projections are trained only for that denoising.

In the pretrained model the written KV already differs from the generation KV by a third of the key norm and 70% of the value norm, later segments attend to the two differently, and swapping the written KV for the generation KV causally degrades the continuation (Sec. 5.3). To let memory writing adapt independently of denoising, we introduce a writer-specific low-rank residual [6] on the shared backbone, active only when a segment is written into the cache, so that the write path has one set of parameters to itself while the backbone stays shared and fully trained (Sec. 4).

Learning such a residual requires the gradient that previous recipes cut off [7, 32, 34]: from future reads back to past writes. Re-enabling that gradient through the ordinary KV cache is not enough. The cache is a mutable buffer written in place inside checkpointed forwards. During backward the replay re-executes those writes and attaches new graph to the same tensor, and training runs out of memory in the first backward pass, even on 5 s clips (Sec. 3.4). Reconstructing the whole clip in a second differentiable forward, as Self Gradient Forcing does [35], avoids the failure, but pays for a second pass and for holding the whole-sequence graph at once. Appendix B.4 gives the code of both alternatives and of dKV Forcing.

We propose *dKV Forcing* (dKV denotes differentiable KV) to compute these future-read gradients within a single temporal traversal, with two execution structures designed for it: persistent differentiable pages and local rematerialization (Figure 2). At training time the inference cache becomes a set of pages: each write commits an immutable page outside any replay region; readers receive the captured prefix through data-free handles, so the graph holds handles while the page data stays outside it; and every graph-carrying forward is rematerialized locally in groups of blocks [2], which bounds the retained activation footprint independently of the persistent state. The traversal computes the exact gradient of the training objective without any second pass (Proposition 1), with 28% less resident memory and 20.8% shorter training cycles than the two-

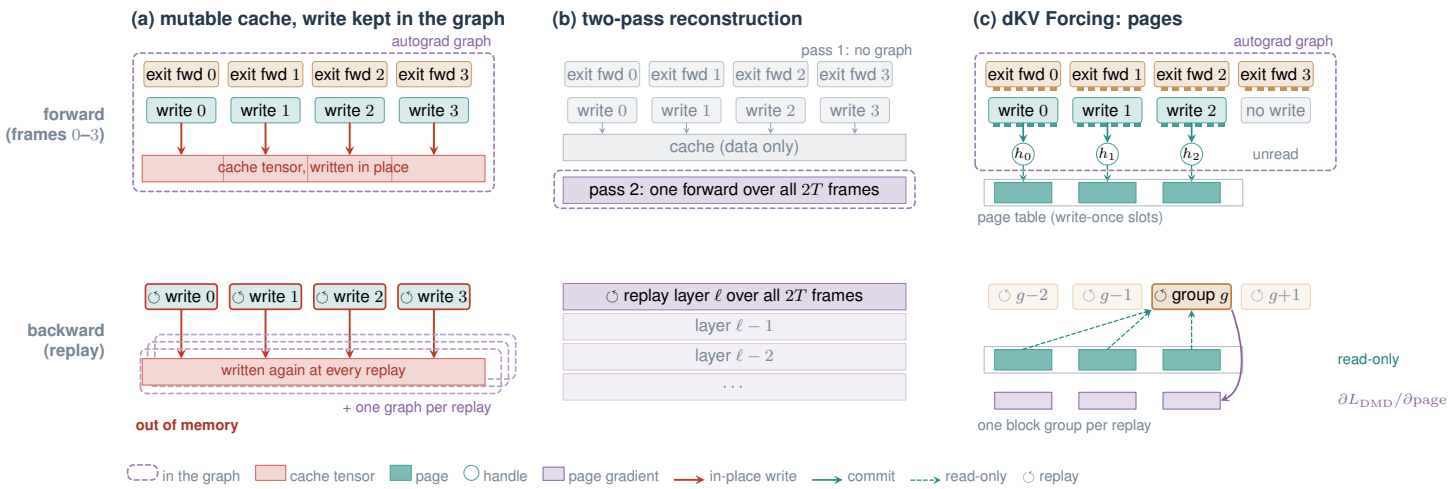


Figure 3. **Autograd graphs and backward replay under three cache execution strategies.** Top: forward computation and retained graphs. Bottom: operations replayed during backward. **(a)** With a mutable cache, writes modify a shared tensor in place. Checkpoint replay repeats these writes, accumulating graphs until backward runs out of memory. **(b)** Two-pass reconstruction builds the rollout without a graph, then recomputes the entire sequence in a differentiable forward, with layer-wise replay during backward. **(c)** dKV Forcing commits each write to an immutable page outside replay. The graph retains handles and checkpoint boundaries, while backward reads the pages and reconstructs one block group at a time.

pass reconstruction (Sec. 5.2). Trained on 5 s clips of the 1.3B Causal Forcing++ generator, the writer residual raises VBench-Long aesthetic quality over SGF, the same objective without writer adaptation, by 6.6 points at 60 s and 8.5 points at 240 s with the highest subject and background consistency (Figure 1), and rollout dropout keeps most of this gain at the motion of detached-write training (Secs. 5.5–5.6).

Our contributions are as follows. (1) *Differentiable memory writing*: a single-traversal training method that carries the exact gradient from future reads back to past KV writes through persistent causal states and local rematerialization. (2) *Writer-specific representation learning*: a low-rank writer residual that learns the memory representation through its future reads while the generation backbone stays shared. (3) *Long-horizon video generation*: gains in quality, consistency and training cost at 60 s and 240 s from 5 s training clips.

2. Background

Notation. A video is a sequence of $T = n + 1$ segments $i = 0, \dots, n$, each one latent frame (frame-wise) or three (chunk-wise), with clean latent x_i. Segment i's denoising state is $z_i^{(t)}$ with $z_i^{(0)} = \epsilon_i$ and $z_i^{(1)} = x_i$ on the grid $0 = t_0 < \cdots < t_k = 1$, and $\tilde{x}_i^{t_j}$ is the clean prediction made at step t_j, the last one being the output x_i. Text conditioning is omitted. Each denoising step predicts the clean latent and re-noises it for the next step:

$$\begin{aligned} \tilde{x}_i^{t_j} &= G_\theta\big(z_i^{(t_j)}, t_j, \mathrm{KV}_{<i}\big), \\ z_i^{(t_{j+1})} &= t_{j+1}\tilde{x}_i^{t_j} + (1 - t_{j+1})\epsilon_{i,j+1}, \qquad x_i = \tilde{x}_i^{t_{k-1}}. \end{aligned} \tag{1}$$

The memory write. After segment i is generated, one more forward writes it into the cache: the same transformer, run on the segment's prediction $\tilde{x}_i^{t_s}$ at the sampled exit step s (Sec. 3.1, the last step at inference), returns the K and V that later segments attend to. As in previous methods [7, 32] the write input is the clean prediction, $\tau = 1$ in

$$x_i^{\mathrm{ctx}} = \tau\tilde{x}_i^{t_s} + (1-\tau)\xi_i, \qquad \mathrm{KV}_i = \mathcal{K}_{\theta,\phi}(x_i^{\mathrm{ctx}}, \tau, \mathrm{KV}_{<i}), \tag{2}$$

where the writer $\mathcal{K}_{\theta,\phi}$ runs the same transformer on x_i^{ctx} and returns the K and V that its self-attention computes, for every layer, and ϕ is the writer residual of Sec. 4, added to those projections. With $\phi = 0$ the write uses the generation projections alone, as in Self Forcing, Causal Forcing++ and SGF. Unlike the K and V computed while denoising, the written K and V are read by every later segment, so the quality of a write is decided by the generations that follow.

The representation hypothesis. Let $W_{a,\ell}$ be the projection matrix of layer ℓ that maps a hidden state to its keys ($a = K$) or values ($a = V$). In the standard design the two roles share the projection weights,

$$W_{a,\ell}^{\mathrm{gen}} = W_{a,\ell}^{\mathrm{mem}}, \qquad a \in \{K, V\}, \tag{3}$$

Algorithm 1 dKV Forcing: single-traversal learning of specialized memory

Require: Parameters θ, ϕ; schedule $\{t_j\}$; context time τ; $T = n + 1$

Sample exit index s and rollout random variables; allocate a page table P with $T - 1$ write-once slots
for $i = 0, \dots, n$ **do**
  $\mathrm{KV}_{<i} \leftarrow$ snapshot of P: pages $0, \dots, i-1$ and handles $h_0, \dots, h_{i-1}$ ▷ Eq. (8)
  $z_i^{(t_0)} \leftarrow \epsilon_i$
  for $j = 0, \dots, s-1$ **do** ▷ no gradient, Eq. (1)
    $z_i^{(t_{j+1})} \leftarrow t_{j+1}\, G_\theta(z_i^{(t_j)}, t_j, \mathrm{KV}_{<i}) + (1 - t_{j+1})\, \epsilon_{i,j+1}$
  end for
  $\bar{z}_i \leftarrow \mathrm{sg}(z_i^{(t_s)})$ ▷ Eq. (4)
  $\tilde{x}_i^{t_s} \leftarrow G_\theta(\bar{z}_i, t_s, \mathrm{KV}_{<i})$ ▷ checkpointed, Eq. (15)
  $\bar{x}_i^{\mathrm{ctx}} \leftarrow \mathrm{sg}(\tau \tilde{x}_i^{t_s} + (1 - \tau)\xi_i)$ ▷ Eqs. (2), (4)
  if $i < n$ **then** ▷ Sec. 3.3
    $\mathrm{KV}_i \leftarrow \mathcal{K}_{\theta,\phi}(\bar{x}_i^{\mathrm{ctx}}, \tau, \mathrm{KV}_{<i})$ ▷ checkpointed
    Commit KV_i to slot i of P; keep handle h_i in the graph
  end if
end for
$g_\theta, g_\phi \leftarrow \nabla_{\theta,\phi} L_{\mathrm{DMD}}(\tilde{x}_0^{t_s}, \dots, \tilde{x}_n^{t_s})$ by reverse mode ▷ Eqs. (10)–(12), (16)
Update θ, ϕ; release P and the graph

where $W_{a,\ell}^{\mathrm{gen}}$ is the projection G_θ uses while denoising and $W_{a,\ell}^{\mathrm{mem}}$ the one $\mathcal{K}$ uses for the write, while the K and V they produce already differ (Sec. 5.3). A projection change $\phi \neq 0$ that helps future readers need not be the one that helps current denoising. Our hypothesis is that letting the writing role specialize its projections improves the downstream utility of what it writes. Learning such a specialization requires the gradient from the future reads of a write back to its writer, which the next section computes in one temporal traversal.

3. Single-traversal learning through persistent causal states

We compute the memory gradient through persistent KV pages and local rematerialization in a single temporal traversal (Figure 2). Sec. 3.1 defines the gradient, Secs. 3.2–3.3 describe its computation, and Sec. 3.4 examines the limitations of mutable caches.

3.1. The objective and its memory gradient

Sample an exit index $s \in \{0, \dots, k-1\}$. Only the exit step and the write are differentiated. The other sampling steps run without gradient, and the write input is detached from the exit prediction:

$$\bar{z}_i = \mathrm{sg}\left(z_i^{(t_s)}\right), \qquad \bar{x}_i^{\mathrm{ctx}} = \mathrm{sg}(x_i^{\mathrm{ctx}}). \tag{4}$$

dKV Forcing differentiates the recurrence of Eqs. (1) and (2) at the exit step under the distribution matching objective $L_{\mathrm{DMD}}(\tilde{X})$ [7, 32] on the exit predictions $\tilde{X} = (\tilde{x}_0^{t_s}, \dots, \tilde{x}_n^{t_s})$, where $\tilde{x}_i^{t_s} = G_\theta(\bar{z}_i, t_s, \mathrm{KV}_{<i})$ is the prediction from $\bar{z}_i$. The gradient of L_{DMD} is

$$\nabla_\theta L_{\mathrm{DMD}} = -\mathbb{E}_{\tilde{X},t,\tilde{X}_t}\big[\big(s_{\mathrm{real}}(\tilde{X}_t, t) - s_{\mathrm{fake}}(\tilde{X}_t, t)\big)\, \partial \tilde{X}/\partial\theta\big], \tag{5}$$

where $\tilde{X}_t = \alpha_t \tilde{X} + \sigma_t \epsilon$ [24, 25], s_{real} is the score of the frozen teacher, and s_{fake} is the score of a critic trained alongside on the generator's own samples. The differential of the exit prediction is

$$\mathrm{d}\tilde{x}_i^{t_s} = D_\theta \tilde{x}_i^{t_s}\, \mathrm{d}\theta + \sum_{j<i} D_{\mathrm{KV}_j} \tilde{x}_i^{t_s}\, \mathrm{dKV}_j. \tag{6}$$

where D_a denotes the Jacobian with respect to a. Treating the history as $\mathrm{sg}(\mathrm{KV}_{<i})$ removes the second term. dKV Forcing preserves this term and the dependencies between successive writes, and its gradient is the exact gradient of this objective (Proposition 1).

The gradient Self Forcing stops. Self Forcing (SF) differentiates the sampled exit-step prediction while stopping gradients through the context write [7], and Causal Forcing++ stops the gradient at the same place [32]: its history is $\mathrm{KV}_i^{\mathrm{SF}} = \mathrm{sg}(\mathcal{K}_\theta(\bar{x}_i^{\mathrm{ctx}}, \tau, \mathrm{KV}_{<i}^{\mathrm{SF}}))$, so $\mathrm{dKV}_j = 0$ in Eq. (6) and, with $v_i = \partial L_{\mathrm{DMD}}/\partial \tilde{x}_i^{t_s}$, its parameter gradient is the first term alone, $\sum_i (D_\theta \tilde{x}_i^{t_s})^{\mathsf{T}} v_i$. SF learns to generate given its self-generated history, while the operations that constructed that history receive no gradient. The second term of Eq. (6) carries the two missing routes: direct use of an earlier write by a later generator, $\theta \to \mathrm{KV}_j \to \tilde{x}_i^{t_s} \to L_{\mathrm{DMD}}$, and use through a later write, $\theta \to \mathrm{KV}_j \to \mathrm{KV}_r \to \tilde{x}_i^{t_s} \to$



Figure 4. **Comparison on 60 s video generation.** Frames at 0–59 s for CF++, SGF and dKV Forcing on two MovieGen prompts, "cat and dog in a sword fight" (top) and "eye with the earth as its iris" (bottom). CF++ loses the scene within the minute and SGF drifts to another layout or breaks up the subject, while dKV Forcing keeps subject, layout and motion. More prompts are in Appendix C.5.

L_{DMD} ($j < r < i$), since dKV_r depends on KV_j through Eq. (2). A residual used only for writing then receives no gradient at all. dKV Forcing, in contrast, preserves both routes, so the gradient reaches the residual (Sec. 3.2).

Gradients through the mutable cache. Enabling gradients through the inference-time KV cache does not work with activation checkpointing. The cache is a mutable buffer written in place by each segment. During backward, checkpoint replay writes to the same buffer again, attaching new autograd graphs while earlier readers still retain references to it. These graphs accumulate and exhaust GPU memory even on 5 s clips (Sec. 3.4, Figure 3a). The mutable buffer also breaks the correctness of replay: checkpointing requires each forward to read its original history [18], which later writes may have changed. Persistent pages avoid both problems by keeping committed history unchanged and separating memory writes from checkpoint replay (see Appendix B.5 for a detailed analysis).

3.2. Persistent causal states and the memory gradient

To meet both requirements and enable the gradient of every past write, we replace the mutable cache by persistent pages. Each write becomes an immutable page that later segments read through a handle, and the history is the ordered tuple of pages: for layer $\ell \in \{1, \dots, B\}$ the writer emits a page $P_{i,\ell} = (K^{\mathrm{mem}}_{i,\ell}, V^{\mathrm{mem}}_{i,\ell})$, its ordinary K and V of that layer with the autograd path preserved, and $\mathrm{KV}_i = (P_{i,1}, \dots, P_{i,B})$. Only writer outputs persist, and generation reads the historical pages together with the transient KV of its own tokens (Appendix A.5). The page table that manages them is given in code in Appendix B.4. Writing $w = (\theta, \phi)$, the history before segment i and its extension by the write of segment i are

$$\begin{aligned} \mathcal{H}_i &= (\mathrm{KV}_0, \dots, \mathrm{KV}_{i-1}), \\ \mathcal{H}_{i+1} &= \mathcal{E}_{i,w}(\mathcal{H}_i) := \mathcal{H}_i \,\|\, \big(\mathcal{K}_{\theta,\phi}(\bar{x}^{\mathrm{ctx}}_i, \tau, \mathcal{H}_i)\big). \end{aligned} \tag{7}$$

The transition never changes earlier entries:

$$\Pi_i \mathcal{H}_r = \mathcal{H}_i \quad (r \geq i), \qquad \Pi_i\big(\mathcal{E}_{i,w}(\mathcal{H}_i)\big) = \mathcal{H}_i, \tag{8}$$

where Π_i is the projection that keeps the first i writes of a history. After vectorizing the state tuples, its differential is

$$\mathrm{d}\mathcal{H}_{i+1} = \begin{bmatrix} I \\ D_{\mathcal{H}_i}\mathrm{KV}_i \end{bmatrix} \mathrm{d}\mathcal{H}_i + \begin{bmatrix} 0 \\ D_w \mathrm{KV}_i \end{bmatrix} \mathrm{d}w. \tag{9}$$

The identity block carries every previous write unchanged, and the lower block transmits its effect on the new write.

Table 1. Training cost on one node (4 H100, batch 2 per GPU, bf16, 21-frame clips; median over steady cycles; max over ranks). Cycle = 5 critic + 1 generator update. "Gen. update" = rollout, gradient-bearing forwards, teacher/critic scoring and backward of one generator update; "critic rollout" = the gradient-free generator rollout of one critic update (its critic loss and backward, 5.2 s, are identical across rows). Bold and underline: best and second best among the methods that carry the memory gradient; CF++ carries none and is listed as the reference.

	time (s)				GPU memory (GiB)		
method	cycle	gen. update	critic rollout	vs. two-pass	gen. update peak	whole-run peak	resident
two-pass reconstruction	122.8	35.7	11.6	–	69.9	75.4	40.7
CF++ (writes detached)	104.7	31.8	8.9	−14.8%	55.9	55.9	40.7
mutable cache (write differentiated)	out of memory in the first generator backward						
dKV Forcing (default)	<u>97.3</u>	<u>33.9</u>	7.0	−20.8%	62.1	62.1	**29.2**
dKV Forcing (fused)	**95.9**	**32.4**	**6.8**	−21.9%	65.6	65.6	**29.2**
dKV Forcing (hidden)	105.9	43.1	<u>6.9</u>	−13.7%	**56.1**	**56.1**	<u>29.3</u>
dKV Forcing (fused+hidden)	100.9	37.0	<u>6.9</u>	−17.8%	<u>58.9</u>	<u>58.9</u>	<u>29.3</u>

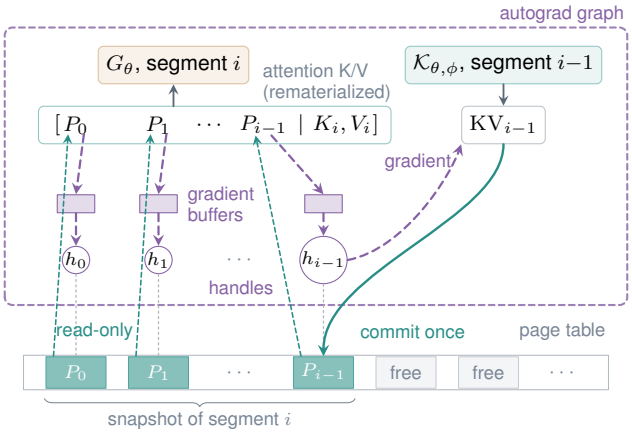


Figure 5. **Page storage and gradient flow in one transformer layer.** The page table (bottom row) stores each write in a separate, write-once slot outside the autograd graph (dashed frame), committed once (teal solid arrow). A data-free handle (purple circle) preserves the gradient connection to the writer without retaining the page data. Each forward reads a snapshot of the visible pages (teal dashed arrows), while backward returns an independent, contiguous gradient buffer (purple rectangles) to each page's writer (purple dashed arrows).

Page table, handles and snapshots. Three structures realize the invariant (Figure 5). Each layer has a contiguous page table P with one slot per segment. The writer output KV_j is copied into its slot P_j once, without gradient, and committed slots are never modified. To preserve the gradient path without retaining another copy of the page data, each writer output KV_j is also connected to a handle h_j in the autograd graph: an expanded zero scalar with the page's shape whose backward passes the incoming gradient unchanged to KV_j. Each forward of segment i receives a snapshot of the table, the visible prefix length i and the handles $h_0, \ldots, h_{i-1}$. Later writes extend the table and leave the visible prefix of earlier snapshots unchanged. A custom autograd operation reads the stored pages $P_0, \ldots, P_{i-1}$ directly in forward and concatenates them with the current segment's K_i, V_i. In backward it splits the incoming gradient by page and routes each slice through h_j to the writer that produced P_j. The slices are copied into separate contiguous buffers so that each page gradient has its own lifetime, independent of the whole history-gradient buffer. The page table itself stays outside the autograd graph. Appendix B.4 gives the code.

The gradient through the writes. Let λ_i be the gradient arriving at the external interface of KV_i. Reversing the recurrence gives

$$\lambda_i = \sum_{r=i+1}^{n} (D_{\mathrm{KV}_i} \tilde{x}_r^{t_s})^\mathsf{T} v_r + \sum_{r=i+1}^{n} (D_{\mathrm{KV}_i} \mathrm{KV}_r)^\mathsf{T} \lambda_r : \quad (10)$$

direct consumption by future generators plus influence through subsequent writes. Because ϕ is used only by the writer,

$$\nabla_\theta L_{\mathrm{DMD}} = \sum_{i=0}^{n} (D_\theta \tilde{x}_i^{t_s})^\mathsf{T} v_i + \sum_{i=0}^{n} (D_\theta \mathrm{KV}_i)^\mathsf{T} \lambda_i, \quad (11)$$

$$\nabla_\phi L_{\mathrm{DMD}} = \sum_{i=0}^{n} (D_\phi \mathrm{KV}_i)^\mathsf{T} \lambda_i, \quad (12)$$

so ϕ is optimized for what future consumers need to read, including effects mediated by later memory formation ($\lambda_n = 0$ without an auxiliary memory loss). The recursion of Eq. (10) can be vectorized:

$$\mathsf{A}_{ri} = \begin{cases} D_{\mathrm{KV}_i} \mathrm{KV}_r, & i < r, \\ 0, & i \geq r, \end{cases} \qquad \eta_i = \sum_{r=i+1}^{n} (D_{\mathrm{KV}_i} \tilde{x}_r^{t_s})^\mathsf{T} v_r, \quad (13)$$

A is strictly block lower triangular with $\mathsf{A}^{n+1} = 0$, so the stacked cotangents $\boldsymbol{\lambda} = (\lambda_0, \ldots, \lambda_n)$ and $\boldsymbol{\eta} = (\eta_0, \ldots, \eta_n)$



Figure 6. **Comparison on 240 s video generation.** Frames at 0–238 s, 46 times the training clip, for CF++, SGF and dKV Forcing on the MovieGen prompt "man wiping a kitchen counter" (top) and the VBench-Long standard prompt "a cat drinking water" (bottom). CF++ has lost the scene, SGF drifts, and dKV Forcing keeps subject, layout and lighting across the full four minutes with the motion intact. More prompts are in Appendix C.5.

satisfy

$$\boldsymbol{\lambda} = (I - \mathsf{A}^{\mathsf{T}})^{-1}\boldsymbol{\eta} = \underbrace{\boldsymbol{\eta}}_{\text{direct use}} + \underbrace{\sum_{q=1}^{n} (\mathsf{A}^{\mathsf{T}})^{q}\boldsymbol{\eta}}_{\text{via later writes}}. \tag{14}$$

The expansion is finite, and reverse mode realizes it through vector–Jacobian products without forming A. Stopping writer-history derivatives removes A and keeps only the direct-use gradient (Appendix C.1).

3.3. Local rematerialization in one temporal traversal

We partition the transformer G_θ or $\mathcal{K}_{\theta,\phi}$ into groups of blocks $\mathcal{B}_g$ and apply activation checkpointing to each group. For role $u \in \{\mathrm{gen}, \mathrm{mem}\}$, let $\mathcal{T}^u_{g,w}$ denote the computation of group g within the generation forward G_θ or the writing forward $\mathcal{K}_{\theta,\phi}$, with parameters w:

$$(h^u_{i,g+1}, \mathcal{P}^u_{i,g}) = \mathcal{T}^u_{g,w}(h^u_{i,g}; \mathcal{H}_i), \tag{15}$$

Here $h^u_{i,g}$ is the hidden state entering group g, and $\mathcal{P}^{\mathrm{mem}}_{i,g}$ contains the pages produced by the writer ($\mathcal{P}^{\mathrm{gen}}_{i,g} = \varnothing$). Only the entry states are kept for backward, and the internal attention and feed-forward activations are recomputed from them. By the time backward runs, the page table holds the final history $\mathcal{H}_n$, and each snapshot still exposes only the prefix that segment i saw in forward:

$$\mathcal{T}^u_{g,w}(h^u_{i,g}; \Pi_i\mathcal{H}_n) = \mathcal{T}^u_{g,w}(h^u_{i,g}; \mathcal{H}_i). \tag{16}$$

Writes are committed outside the checkpointed regions, so replay never modifies a committed page. With parameters and randomness fixed, each group recovers its original outputs and local Jacobians.

Proposition 1 (Exact single-traversal differentiation)**.** *Fix the recorded inputs, model parameters, visibility, positions and random choices across forward execution and backward replay. Assume that each role computation is a pure function of its explicit inputs, state extension preserves the historical prefix and its differential (Eqs. (8) and (9)), and replay satisfies Eq. (16). Then Algorithm 1 computes the same predictions and parameter gradients as the uncheckpointed recurrence of Sec. 3.1, in exact arithmetic. Its memory gradient is given by Eq. (14).*

Table 2. Generation and memory KV differ in representation and in future attention readout: the written K/V against the frame's own generation K/V (same frame, same layer; layer means over 30 layers, 48 videos × 21 frames, ranges over the four checkpoints). "Readout" swaps only frame i's cache entry while every later query is held fixed.

	$\mathrm{KV}^{\mathrm{mem}}$ vs. $\mathrm{KV}^{\mathrm{gen}}$		readout by later frames	
	K	V	attention mass	read-out vector
relative difference $\|A-B\|/\|B\|$	0.35–0.37	0.70–0.72	1.17–1.40×	0.65–0.68
token-wise cosine	0.93	0.73	–	0.76–0.77
per-head linear CKA	0.89–0.91	0.72–0.74	–	–

Appendix A derives the causal factorization and proves the replay equivalence.

Execution and storage. A single traversal is one autoregressive forward sweep with no second whole-sequence reconstruction (Algorithm 1). Each nonterminal segment computes a gradient-bearing exit prediction and a memory write, the final write is omitted, and the other sampling steps run without gradient. In backward, internal activations are rematerialized in groups of five transformer blocks. Let B be the number of transformer blocks and b the group size, and let S be the all-layer KV storage of one segment, H the entry state kept per group, Q the retained prediction and W_b the transient attention workspace per segment of history when one group of b blocks is recomputed and backpropagated, live for one group at a time. The retained tensor memory for a clip of T segments is approximately

$$O(TS + 2T\,\lceil B/b\rceil\,H + TQ + TW_b)\,, \qquad (17)$$

excluding parameters, optimizer state, score models and metadata. The four terms are the pages (TS, including their cotangents up to a constant), the entry states of the exit and write forwards of every segment ($2T\lceil B/b\rceil H$), the retained predictions (TQ) and the workspace of the one group under recomputation (TW_b). Every term is linear in T, the group size b trades the entry states against the workspace, and attention over the full history keeps its quadratic cost (Appendix B.5).

3.4. Mutable cache versus pages

Memory retention under checkpoint replay. The storage bound of Eq. (17) relies on persistent state that stays outside local replay. In the mutable cache, the replayed writes of Sec. 3.1 attach new graph connections (`CopySlices`) to the externally retained tensor, so replay graphs outlive their use. The saved-input closure also retains a nested reference to the cache. In a full-size generator-only measurement, isolating the replay-time cache writes lowers the first backward peak from 57.22 to 28.71 GiB, within 0.6 GiB of pages (Appendix B.5).

Retained state of each scheme. Figure 3 compares the three executions. Each rollout frame uses five forwards: four denoising forwards, including the sampled exit that feeds the loss, and one write forward. The schemes differ in which forwards carry a graph, how historical K/V enter it and when activations are released. *Self Forcing* keeps only the exit-forward graph, with the write detached. *Differentiating the mutable cache directly* runs out of memory under checkpoint replay (Sec. 3.1), and without checkpointing before frame six. *The two-pass reconstruction* records exit inputs and clean contexts during a no-grad rollout, then recomputes context and exit forwards in one parallel teacher-forced pass over all $2T$ frames. Historical K/V are ordinary outputs, with no in-place writes. It retains the whole second-pass forward's checkpoint boundaries and attention working set, replays it layer by layer in backward, and keeps the T-frame cache allocated between steps. *dKV Forcing* allocates the page table once and keeps the exit and write graphs within the rollout. The write input is detached, and new K/V are committed to immutable pages outside replay, with data-free handles preserving graph connectivity. Across frames, page data, group entry states and the exit predictions persist, and internal activations are rematerialized in groups of five blocks. In backward each page receives its own contiguous gradient buffer (Sec. 3.2), and the store releases the page table after backward. The default execution retains the $(T-1)$ committed pages and their cotangents, and the hidden-state variants retain the writer's input instead (Table 1). Table 1 measures the costs, and Figure 8 in Appendix B.5 gives the measured memory composition of one generator update, classified by allocation site.

4. Separating generation and memory representations

With gradients from future reads available, we introduce writer-specific adaptation to learn a memory representation distinct from generation: a low-rank residual on the K and V projections, active only when a segment is written into the cache, relaxes the shared-projection constraint of Eq. (3) at the writer and leaves the generation path unchanged.

The generator and writer process different latents and

time embeddings, so their hidden states $h^{\text{gen}}_{i,\ell}$ and $h^{\text{mem}}_{i,\ell}$ already differ. What the architecture ties is the projection parameters. At layer ℓ the two roles use

$$\begin{aligned} K^{\text{gen}}_{i,\ell} &= \mathcal{R}_i\Big(\mathcal{N}_K(W_{K,\ell}h^{\text{gen}}_{i,\ell})\Big), \qquad V^{\text{gen}}_{i,\ell} = W_{V,\ell}h^{\text{gen}}_{i,\ell}, \\ K^{\text{mem}}_{i,\ell} &= \mathcal{R}_i\big(\mathcal{N}_K((W_{K,\ell}+\Delta W_{K,\ell})h^{\text{mem}}_{i,\ell})\big), \\ V^{\text{mem}}_{i,\ell} &= (W_{V,\ell}+\Delta W_{V,\ell})h^{\text{mem}}_{i,\ell}, \end{aligned} \tag{18}$$

with $\mathcal{N}_K$ the key normalization, $\mathcal{R}_i$ the positional rotation and $\Delta W_{a,\ell} = U_{a,\ell}Z_{a,\ell}$ the low-rank construction of Hu et al. [6] (rank 64, U zero-initialized, Z random; biases unchanged). Only these residual parameters form ϕ. The untying is partial and follows the computational role: generation-side KV stays transient within a denoising evaluation, writer-side KV becomes the historical context. The backbone keeps training. For fixed θ and a fixed historical input a generation call depends on ϕ only through previous writes. We introduce writer-specific adaptation as a lightweight way to allocate learnable capacity to memory construction. Sec. 5.4 evaluates its contribution on top of differentiable memory training and the effect of writer capacity. Inference uses the ordinary cache with the residual enabled at cache refresh and disabled during denoising. The same execution supports attenuated page gradients and an independent writer (Appendix C.1).

5. Experiments

Section 5.2 measures gradient agreement and cost, and Secs. 5.3–5.6 measure the written representation, the effect of writer parameters, the benchmarks and the fidelity–motion trade-off. Appendix B records every run.

5.1. Setup

All runs fine-tune the 1.3B causal generator of Causal Forcing++ [32] (Wan2.1-T2V-1.3B) with its asymmetric video-level DMD objective and the frozen 14B teacher [24, 25], starting from its released frame-wise and chunk-wise causal consistency-distillation (CD) initializations: 1500 generator updates and a global batch of 8 clips of 21 latent frames ($\approx$ 5 s) from the extended VidProM prompts. Training attends to the full 21-frame history. The rank-64 writer residual acts on the K and V projections of every self-attention layer (11.8M parameters, 0.9% of the generator, used only by the write forward) with $U = 0$, so the adapted model reproduces the released checkpoint at step 0. Appendix B.1 gives the remaining training and inference settings.

CF++ is the original Causal Forcing++ training path with detached writes, as in Self Forcing, run from the same code base. *SGF* is the memory-gradient objective of Self Gradient Forcing [35] trained with our single traversal, that is, dKV Forcing without the writer residual (Sec. 5.2 verifies that the two implementations compute the same gradient). It is the primary baseline: it already trains memory through future use, so gains from the writer residual (Table 4) are measured against a model that already receives the memory gradient. All configurations share the released CF++ initialization and recipe, the data and batch order, the schedule and matched seeds, which isolates memory-gradient training and writer adaptation. *Writer LoRA* is SGF with the writer residual. The configuration marked *(ours)*, labelled dKV Forcing in the figures, trains the residual with rollout dropout 0.5 (Sec. 5.6).

Evaluation uses VBench at 5 s [8] and VBench-Long at 60 s and 240 s (46 times the training clip), reporting subject and background consistency, aesthetic and imaging quality and dynamic degree. Results are a mean $\pm$ SD over training seeds (n per row), with Tables 5 and 6 on the same three seeds for every configuration (details in Appendix B.1).

5.2. Single traversal versus two-pass reconstruction

Correctness. The single traversal computes the fixed-recorded-input gradient of a two-pass reconstruction of the same objective (Proposition 1). In fp64 the relative gradient error of all four executions of Table 1 against the two-pass reference is at most 4.1×10^{-12}, and the default execution is bitwise identical to the pre-optimization traversal in full-size bf16 and multi-step trainer checks. For every checkpoint evaluated here the page training path, the ordinary cache path and the streaming inference path agree to rounding at 21 frames. Appendix B.3 reports the numerical and distributed validation, including chunk-wise generation and the fused and hidden-state variants.

Time and memory. Table 1 compares, in the production setting (4 GPUs, batch 2 per GPU, bf16, FSDP), the implementations of the same fixed-input gradient target together with CF++, which carries no memory gradient. The two-pass reconstruction is the public implementation of Self Gradient Forcing [35], a rollout with detached writes followed by a second whole-sequence differentiable forward, and the mutable cache with the write differentiated is the CF++ implementation with the stop-gradient on its cache write removed (Sec. 3.4). The dKV Forcing rows are the single traversal of Sec. 3 (default) and two further optimizations of it, implemented as described in Appendix B.2: a fused write that batches the exit and write roles after an untracked exit evaluation (fused), and hidden-state pages that store hidden states as pages and rebuild K/V from them (hidden, and fused+hidden with both). Costs are measured over steady training cycles of five critic updates and one generator update.

The full memory gradient adds 7% to the generator-update time of detached-write training (33.9 against 31.8 s). Against the two-pass reconstruction of the same gradient, the default execution lowers the generator-update peak from

Table 3. Replacing written KV changes the autoregressive continuation. With checkpoint, prompt and sampling randomness fixed, the written K/V of frames 0–2 or of all 21 frames is replaced by their generation K/V at 5 s (48 prompts per checkpoint). Next-frame change: latent relative L_2 to the unmodified rollout, with an independent seed in parentheses as the reference scale. Δ: paired change of the prefix→later DINO consistency (mean $\pm$ s.e.m.).

checkpoint	next-frame change (other seed)	Δ consistency, frames 0–2	Δ consistency, all frames
CD initialization (untrained)	0.55 (0.97)	-0.231 ± 0.024	-0.597 ± 0.018
SGF	0.57 (1.08)	-0.059 ± 0.010	-0.183 ± 0.017
writer LoRA, $\alpha = 1$	0.56 (1.07)	-0.070 ± 0.009	-0.166 ± 0.014
writer LoRA, $\alpha = 0.5$	0.57 (1.05)	-0.073 ± 0.008	-0.166 ± 0.019

Table 4. Disentangling the effects of differentiable memory writing and writer adaptation: VBench-Long 60 s, step 1500, frame-wise, mean $\pm$ SD over three to five training seeds. "Residual" says where a rank-64 residual is active; "memory gradient" says which parameters receive gradient from future reads. The dedicated writer row replaces the residual by a fully separate writer network (three seeds). *Our most balanced configuration: the writer residual with rollout dropout 0.5; the next row is the same residual without dropout. Best value per column among the first four rows in bold, second best underlined.

configuration	memory gradient	residual	aesthetic	dynamic	imaging	subject
CF++, detached writes	none	none	51.6 ± 2.4	$\mathbf{74.9} \pm 15.8$	62.7 ± 1.8	94.8 ± 1.1
SGF	backbone	none	57.2 ± 2.6	38.7 ± 16.3	68.0 ± 1.8	$\underline{96.7} \pm 0.8$
writer LoRA + rollout dropout 0.5 (ours)*	backbone + residual	write	$\underline{60.1} \pm 1.2$	$\underline{74.0} \pm 28.7$	$\underline{69.0} \pm 1.1$	96.6 ± 1.0
writer LoRA without rollout dropout	backbone + residual	write	$\mathbf{62.1} \pm 1.4$	22.6 ± 14.4	$\mathbf{69.6} \pm 0.6$	$\mathbf{97.9} \pm 0.6$
separate writer network (second 1.3B copy, backbone learning rate)	backbone + writer network	none (full network)	64.9 ± 0.6	26.0 ± 18.2	71.6 ± 0.7	98.6 ± 0.4

69.9 to 62.1 GiB and, because pages live only for one traversal, resident memory by 28% (29.2 against 40.7 GiB, the value that CF++'s preallocated cache also holds). The page execution also serves the gradient-free rollouts of the critic updates, which run faster on pages than on the mutable cache or in the two-pass code, and these rollouts account for most of the 20.8% shorter training cycle (97.3 against 122.8 s, and 7% shorter than CF++). The mutable cache with the write differentiated runs out of memory in every setting. The fused write lowers the cycle time further at a higher peak, and hidden-state pages give the lowest peaks at more computation. With the writer residual, the costs change by less than 1 s and 0.1 GiB.

5.3. Generation KV and memory KV are different representations

For a frame i and layer ℓ we compare $\mathrm{KV}^{\mathrm{gen}}_{i,\ell}$, the K/V the frame used in its own last denoising evaluation, with $\mathrm{KV}^{\mathrm{mem}}_{i,\ell}$, the K/V written at cache refresh from its clean prediction (Table 2). The two come from the same backbone projections and different inputs, a noisy latent and the clean prediction. Across the evaluated checkpoints they differ by about a third of the key norm and 70% of the value norm, with only partly shared geometry (per-head CKA 0.90 for K and 0.73 for V). The gap is already present in the pretrained model and changes little after memory-gradient training without writer adaptation (Appendix C.2). Later frames also read the two differently: with every query fixed, a frame represented by its written K/V attracts 1.17–1.40× the attention mass it attracts as generation K/V, and the read-out vector changes by two thirds of its norm, so the difference changes what future frames retrieve from the same history.

The written representation matters causally. With checkpoint, prompt, noises and schedule fixed, we replace the memory of the first three frames by their generation K/V and let the rollout continue (Table 3 and Figure 9 in Appendix C.2). The next frame moves by about half the distance between two independent seeds (latent relative L_2 of 0.55–0.57 against 0.97–1.08), and the DINO similarity of later frames to the opening drops by 0.06–0.23 in every checkpoint, monotonically in the number of intervened frames. The deficit persists through 60 s rollouts. The same intervention at frames 11–13 has a smaller latent effect and no consistency effect: the continuation depends on what the opening frames wrote. Controls and per-checkpoint results are in Appendix C.2.

5.4. Effects of memory gradients and writer adaptation

Table 4 separates the effects of memory-gradient training and writer-specific adaptation at 60 s, with three to five training seeds per row. Every row trains all 1.3B backbone parameters under the same initialization, objective and budget, and the residual rows add 0.9% parameters that only the write forward uses, so the comparison isolates where the write can adapt. The memory gradient alone (CF++→SGF) is worth 5.6 points of aesthetic quality while dynamic degree falls from 75 to 39. Writer-specific parameters add a

Table 5. Main results at 5 s and 60 s, mean ± SD over training seeds $\{1, 10, 20\}$, step 1500. 5 s: official VBench Total / Quality / Semantic scores and dynamic degree. 60 s: VBench-Long standard prompts (aesthetic, dynamic, imaging, subject, background). Results at 240 s are in Table 6. Per block, best value per column in bold, second best underlined.

	VBench 5 s				VBench-Long 60 s				
configuration	Total	Quality	Semantic	dynamic	aesthetic	dynamic	imaging	subject	backgr.
frame-wise (one latent frame per segment)									
CF++	**83.5** ± 0.7	**84.8** ± 0.5	78.1 ± 1.5	**84.0** ± 12.9	52.7 ± 1.4	**76.5** ± 19.0	63.3 ± 1.6	94.7 ± 1.3	94.5 ± 0.7
SGF	83.1 ± 0.4	84.1 ± 0.3	79.4 ± 1.1	59.4 ± 7.6	56.2 ± 1.4	38.5 ± 6.9	67.4 ± 2.2	96.5 ± 0.4	95.7 ± 0.5
writer LoRA	83.1 ± 1.0	83.8 ± 1.2	**80.4** ± 0.2	36.7 ± 19.6	**62.8** ± 0.4	21.8 ± 17.5	**69.7** ± 0.6	**98.2** ± 0.2	**96.9** ± 0.2
+ rollout dropout 0.5	82.9 ± 0.1	83.8 ± 0.2	79.3 ± 0.9	76.2 ± 28.1	59.8 ± 1.3	71.4 ± 34.5	68.8 ± 1.3	96.6 ± 1.2	94.8 ± 1.1
chunk-wise (three latent frames per segment)									
CF++	83.8 ± 0.1	84.8 ± 0.2	**79.9** ± 0.5	69.1 ± 10.8	57.8 ± 1.6	50.6 ± 15.8	65.6 ± 1.2	96.8 ± 0.7	95.7 ± 0.7
SGF	83.9 ± 0.2	85.0 ± 0.2	79.5 ± 0.4	71.3 ± 11.6	58.1 ± 0.1	48.4 ± 19.8	67.8 ± 1.4	**96.9** ± 0.5	95.7 ± 0.6
writer LoRA	**84.2** ± 0.5	**85.4** ± 0.5	79.1 ± 0.8	**80.7** ± 4.8	**59.6** ± 0.8	**61.9** ± 19.7	**68.7** ± 1.7	96.8 ± 0.8	**95.8** ± 0.1

Table 6. 240 s results, VBench-Long standard prompts with 963 latent frames (four times the 60 s horizon), mean ± SD over training seeds $\{1, 10, 20\}$, step 1500, except rollout dropout, whose entire row uses the two-seed subset marked in the table. Per block, best value per column in bold, second best underlined.

configuration	aesthetic	dynamic	imaging	subject	background	smoothness	flicker
frame-wise							
CF++	49.0 ± 2.5	**81.7** ± 14.9	61.7 ± 2.7	93.8 ± 1.0	94.4 ± 0.7	97.5 ± 1.5	98.4 ± 0.3
SGF	51.3 ± 1.7	38.9 ± 7.4	64.6 ± 3.5	95.9 ± 0.3	95.3 ± 0.4	98.6 ± 0.2	98.0 ± 0.7
writer LoRA	**59.8** ± 1.4	20.5 ± 19.2	67.7 ± 1.3	**97.7** ± 0.3	**96.4** ± 0.3	**99.0** ± 0.2	**99.2** ± 0.3
+ rollout dropout 0.5	57.4 ± 0.4	72.0 ± 33.8	**68.2** ± 0.5	96.2 ± 1.0	94.8 ± 0.7	97.7 ± 0.9	98.5 ± 0.1 (n=2)
chunk-wise							
CF++	52.5 ± 4.3	50.4 ± 16.1	62.3 ± 2.9	95.9 ± 0.5	95.3 ± 0.4	98.0 ± 0.7	98.7 ± 0.6
SGF	53.2 ± 1.0	46.8 ± 20.4	65.4 ± 2.7	96.4 ± 0.5	**95.4** ± 0.5	98.6 ± 0.5	98.7 ± 0.3
writer LoRA	**55.7** ± 2.2	**57.1** ± 24.3	**66.2** ± 2.3	**96.6** ± 0.6	**95.4** ± 0.1	**98.8** ± 0.3	**98.9** ± 0.2

second gain on top of it: the residual raises aesthetic quality by a further 4.9 points, at a dynamic degree of 23 ± 14. Rollout dropout recovers the motion: with the residual disabled for half of the training rollouts, dynamic degree returns to 74 while the model keeps 2.9 points of aesthetic quality over SGF. A dedicated writer network, a second 1.3B copy that computes every write and is trained only through the future use of those writes, raises aesthetic quality to 64.9 ± 0.6, another 2.8 points beyond the residual without dropout: writer-specific capacity keeps paying on top of the full memory gradient (Appendix C.1).

5.5. Benchmark results

Tables 5 and 6 report the production configurations at 5, 60 and 240 s with the same three seeds (the marked dropout row at 240 s excepted). We report two operating points of the same model: the writer residual, its most consistent point, and the residual with rollout dropout 0.5, our balanced default (starred in Table 4). At the 5 s training horizon every configuration keeps VBench Total within 82.9–84.2, so the long-horizon gains come at no short-horizon cost. The gains widen with the horizon: frame-wise, the residual raises aesthetic quality over SGF by 6.6 points at 60 s and 8.5 points at 240 s, 46 times the training clip, with the highest subject and background consistency in both tables (Figures 4 and 6). Rollout dropout combines the two: at 60 s it reaches 59.8 points of aesthetic quality, 3.6 above SGF, at the motion of the detached-write recipe (71 ± 35 against 77), and at 240 s it improves aesthetic quality over SGF by 6.1 points at a dynamic degree of 72 ± 34. Chunk-wise, the residual improves aesthetic quality over SGF by 1.5 points at 60 s and 2.5 at 240 s with a higher mean dynamic degree and consistency within seed variation of SGF. Sec. 5.6 examines the fidelity–motion trade-off.

5.6. Controlling fidelity and motion

Differentiable memory training raises long-horizon fidelity and, in frame-wise generation, trades motion for it (Figure 7). The trade-off follows the memory gradient. On the real DMD gradient of a trained model the memory part is as large as the direct part and nearly orthogonal to it ($|g_{\text{mem}}|/|g_{\text{direct}}| = 1.0$–$1.1$, cosine -0.08 to -0.14; Appendix C.1), and scaling it by α at each writer-output interface moves SGF monotonically from the detached write

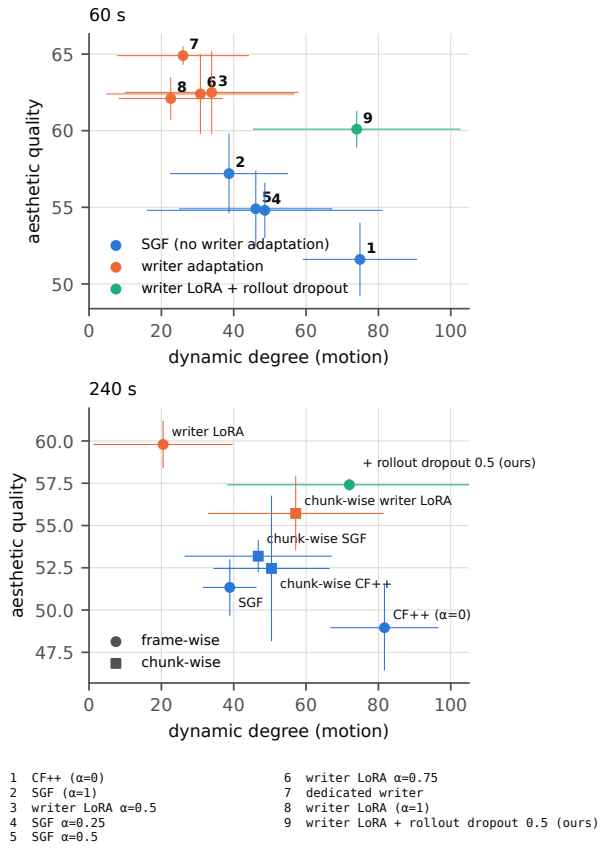


Figure 7. Fidelity versus motion. Top: the frame-wise configurations of Secs. 5.4 and 5.6 at 60 s (VBench-Long standard prompts; mean ± SD over training seeds; numbers index the key below the plot). Bottom: the reported frame-wise and chunk-wise configurations at 240 s (seeds 1, 10, 20, except the dropout row marked in Table 6).

(51.6 aesthetic, 75 dynamic) to the full gradient (57.2, 39; Table 12 in Appendix C.1). The writer residual extends the curve (62.1, 23) and makes its gain robust to the scale: $\alpha = 0.5$ and 0.75 stay within seed variation of $\alpha = 1$. Gradient scaling sets the strength of the memory-gradient paths, whereas rollout dropout decides whether the residual takes part in a training rollout, and the latter is what moves motion. Rollout dropout, which disables the residual for an entire training rollout with probability p, trains the backbone to generate without the residual for half of the rollouts and brings the motion back with it: at $p = 0.5$ dynamic degree returns to 74 ± 29 at 60.1 ± 1.2 points of aesthetic quality, three above SGF, and the same operating point holds at 240 s (57.4 / 72 against 51.3 / 39 for SGF). The differences between the ends of the curve and between the dropout point and SGF exceed the seed spread of dynamic degree, and the cache intervention of Sec. 5.3 shows that the written representation carries the motion effect.

6. Related work

Self-rollout training and context gradients. Self Forcing trains autoregressive video diffusion on generated histories with a video-level objective and detached writes [7]. Causal Forcing introduces a causal teacher and causal ODE distillation [34], and Causal Forcing++ uses causal consistency distillation and an asymmetric DMD objective [32]. Both retain detached memory writes. Self Gradient Forcing [35] restores context gradients through a detached rollout followed by a second, whole-sequence differentiable forward. We compute the same gradient target in a single temporal traversal and use its two-pass implementation as the correctness and cost reference of Sec. 5.2. Concurrent Connected Self Forcing [28] reconnects gradients across chunks through the generated latents with shortcut gradient replay.

Role-specific representations and adaptation. Parameter-efficient adaptation introduces trainable modules into shared backbones: adapters [5], prefix and prompt tuning [13, 14], and low-rank residuals [6] with adaptive or decomposed variants [17, 30]. We use low-rank residuals to specialize the K/V projections for memory writing while the generation backbone stays shared. Cache compression and eviction methods [15, 21, 31] select which historical entries remain available at inference, and PagedAttention [12] manages the physical storage of inference-time KV blocks. Our differentiable pages instead preserve gradients from future reads to historical writes during training, while inference uses the ordinary cache with its attention sink [21]. Long-horizon causal generators extend their context through rolling-window denoising [16], streaming tuning with KV re-caching [23], packed frame context [29] and retrieved memories [22, 27]. Our approach learns the representation written into that context.

Rematerialization. Checkpointing schedules for reverse-mode differentiation [3], memory-efficient backpropagation through time [4, 20], sublinear-memory training [2], optimal and dynamic tensor rematerialization [9, 10] and selective recomputation in large transformers [11] reduce activation storage through recomputation. We use activation checkpointing [2, 18] and sharded data parallelism [33] while keeping committed KV pages outside the checkpointed computations, so that backward replay reconstructs each local computation from its original historical prefix with the gradients to earlier writes preserved through the pages (Sec. 3.4).

7. Conclusion

We presented dKV Forcing, which learns a dedicated memory representation from the gradients of its future reads. A

writer-specific residual separates memory writing from generation on the shared backbone, and persistent differentiable KV pages with local rematerialization compute these gradients in a single temporal traversal, at a lower memory and training-cycle cost than the two-pass reconstruction. Trained on 5 s clips, the model holds subject, layout and lighting for four minutes with the highest consistency and aesthetic quality among the recipes compared, and rollout dropout keeps most of that gain at the motion of detached-write training. The pages and the residual are not tied to video: any causal cache whose writes are read later can be trained the same way.

Two limitations still exist. Our experiments focus on a 1.3B causal video generator trained on 5 s clips. We plan to extend dKV Forcing to other causal architectures, particularly action-conditioned video generation. We also observe a trade-off between long-horizon fidelity and motion. While we have explored this trade-off empirically, understanding its underlying causes and developing writing objectives that adapt to different prompts or segments remain important directions for future work.

References

[1] Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. In *Advances in Neural Information Processing Systems*, 2024. 2

[2] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016. 2, 12

[3] Andreas Griewank and Andrea Walther. Algorithm 799: revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software*, 26(1):19–45, 2000. 12

[4] Audrunas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. Memory-efficient backpropagation through time. In *Advances in Neural Information Processing Systems*, 2016. 12

[5] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, 2019. 12

[6] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. 2, 9, 12

[7] Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion. *arXiv preprint arXiv:2506.08009*, 2025. 2, 3, 4, 12

[8] Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, Yaohui Wang, Xinyuan Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. VBench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. 9, 16

[9] Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gholami, Pieter Abbeel, Joseph Gonzalez, Kurt Keutzer, and Ion Stoica. Checkmate: Breaking the memory wall with optimal tensor rematerialization. In *Machine Learning and Systems*, 2020. 12

[10] Marisa Kirisame, Steven Lyubomirsky, Altan Haan, Jennifer Brennan, Mike He, Jared Roesch, Tianqi Chen, and Zachary Tatlock. Dynamic tensor rematerialization. In *International Conference on Learning Representations*, 2021. 12

[11] Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. In *Machine Learning and Systems*, 2023. 12

[12] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023. 12

[13] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Conference on Empirical Methods in Natural Language Processing*, 2021. 12

[14] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Annual Meeting of the Association for Computational Linguistics*, 2021. 12

[15] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, 2024. 12

[16] Kunhao Liu, Wenbo Hu, Jiale Xu, Ying Shan, and Shijian Lu. Rolling forcing: Autoregressive long video diffusion in real time. *arXiv preprint arXiv:2509.25161*, 2025. 12

[17] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *International Conference on Machine Learning*, 2024. 12

[18] PyTorch Contributors. torch.utils.checkpoint. Official PyTorch documentation, 2026. Accessed September 7, 2026. 5, 12, 16

[19] Wan Team. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025. 1

[20] Ronald J. Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 2(4):490–501, 1990. 12

[21] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024. 12

[22] Zeqi Xiao, Yushi Lan, Yifan Zhou, Wenqi Ouyang, Shuai Yang, Yanhong Zeng, and Xingang Pan. WorldMem: Long-term consistent world simulation with memory. *arXiv preprint arXiv:2504.12369*, 2025. 12

[23] Shuai Yang, Wei Huang, Ruihang Chu, Yicheng Xiao, Yuyang Zhao, Xianbang Wang, Muyang Li, Enze Xie, Yingcong Chen, Yao Lu, Song Han, and Yukang Chen. LongLive: Real-time interactive long video generation. *arXiv preprint arXiv:2509.22622*, 2025. 12

[24] Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Fredo Durand, and William T. Freeman. Improved distribution matching distillation for fast image synthesis. *arXiv preprint arXiv:2405.14867*, 2024. 2, 4, 9

[25] Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Fredo Durand, William T. Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. 2, 4, 9

[26] Tianwei Yin, Qiang Zhang, Richard Zhang, William T. Freeman, Frédo Durand, Eli Shechtman, and Xun Huang. From slow bidirectional to fast autoregressive video diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. 2

[27] Jiwen Yu, Jianhong Bai, Yiran Qin, Quande Liu, Xintao Wang, Pengfei Wan, Di Zhang, and Xihui Liu. Context as memory: Scene-consistent interactive long video generation with memory retrieval. *arXiv preprint arXiv:2506.03141*, 2025. 12

[28] Dongbin Zhang, Chaoda Zheng, Kangjie Chen, Xiangyu Li, Shijia Chen, Jinhao Deng, Yuqi Zhang, Guangfeng Jiang, Hongbin Lin, Choo Sin Wai, Minqi Wang, Puyi Wang, Jingye Zhang, Yu Zhang, Xianming Liu, and Boyang Wang. Connected self forcing: Beyond local learning in video autoregression. *arXiv preprint arXiv:2610.12156*, 2026. 12

[29] Lvmin Zhang and Maneesh Agrawala. Packing input frame context in next-frame prediction models for video generation. *arXiv preprint arXiv:2504.12626*, 2025. 12

[30] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*, 2023. 12

[31] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, 2023. 12

[32] Min Zhao, Hongzhou Zhu, Kaiwen Zheng, Zihan Zhou, Bokai Yan, Xinyuan Li, Xiao Yang, Chongxuan Li, and Jun Zhu. Causal forcing++: Scalable few-step autoregressive diffusion distillation for real-time interactive video generation. *arXiv preprint arXiv:2605.15141*, 2026. 2, 3, 4, 9, 12

[33] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. PyTorch FSDP: Experiences on scaling fully sharded data parallel. *Proceedings of the VLDB Endowment*, 16(12), 2023. 12

[34] Hongzhou Zhu, Min Zhao, Guande He, Hang Su, Chongxuan Li, and Jun Zhu. Causal forcing: Autoregressive diffusion distillation done right for high-quality real-time interactive video generation. *arXiv preprint arXiv:2602.02214*, 2026. 2, 12

[35] Junhao Zhuang, Shiyi Zhang, Yuxuan Bian, Yaowei Li, Yawen Luo, Yijun Liu, Weiyang Jin, Songchun Zhang, Xianglong He, Xuying Zhang, Haoran Li, Haoyang Huang, Zeyue Xue, and Nan Duan. Self gradient forcing: Native long video extrapolation. *arXiv preprint arXiv:2607.20368*, 2026. 2, 9, 12, 24

dKV Forcing: Separating Generation and Memory Representations through a Differentiable KV Cache

Supplementary Material

Contents of the appendix

A Derivations and correctness **15**
- A.1. Finite causal factorization of the memory gradient . . . 15
- A.2. Proof of Proposition 1 . . . 15
- A.3. Two distinct temporal routes . . . 16
- A.4. Role-specificity of the writer residual . . . 16
- A.5. Gradients through the mutable cache . . . 16

B Implementation and experimental details **16**
- B.1. Experimental setup . . . 16
- B.2. Implementation . . . 17
- B.3. Validation . . . 17
- B.4. Page execution in PyTorch . . . 18
- B.5. Performance measurements . . . 20

C Additional experiments and results **21**
- C.1. Controlling the memory gradient and writer parameters . . . 21
- C.2. KV representation analyses and intervention controls . . . 23
- C.3. Fidelity versus motion . . . 24
- C.4. Complete results . . . 24
- C.5. Additional qualitative comparisons . . . 24

A. Derivations and correctness

A.1. Finite causal factorization of the memory gradient

Vectorize and stack all memory outputs as $\mathbf{k} = \mathrm{col}(\mathrm{vec}\,\mathrm{KV}_0, \ldots, \mathrm{vec}\,\mathrm{KV}_n)$, where the terminal write KV_n is a formal node with zero cotangent that Algorithm 1 omits. With the recorded inputs fixed, each write KV_i depends on the parameters w and on the earlier writes $\mathrm{KV}_{<i}$. Stacking the writes gives the causal recurrence $\mathbf{k} = \mathbf{W}_w(\mathbf{k})$, whose component i uses only components $j < i$. Let $\mathsf{A} = D_{\mathbf{k}}\mathbf{W}_w$ and $\mathsf{B}_w = D_w\mathbf{W}_w$, with historical inputs held independent when taking the local parameter derivative. Linearizing gives

$$\mathrm{d}\mathbf{k} = \mathsf{A}\,\mathrm{d}\mathbf{k} + \mathsf{B}_w\,\mathrm{d}w, \qquad \mathrm{d}\mathbf{k} = (I-\mathsf{A})^{-1}\mathsf{B}_w\,\mathrm{d}w. \tag{19}$$

Causality makes A strictly block lower triangular. Every product of q blocks corresponds to a chain of q increasing segment indices. No such chain has $n+1$ edges, so $\mathsf{A}^{n+1} = 0$ and

$$(I-\mathsf{A})^{-1} = \sum_{q=0}^{n} \mathsf{A}^q. \tag{20}$$

For the stacked generation map $\mathbf{F}_\theta(\mathbf{k})$ and output cotangent $\mathbf{v}$, define $\boldsymbol{\eta} = (D_{\mathbf{k}}\mathbf{F}_\theta)^{\mathsf{T}}\mathbf{v}$. Then

$$\mathrm{d}L = \mathbf{v}^{\mathsf{T}} D_w\mathbf{F}_\theta\,\mathrm{d}w + \boldsymbol{\eta}^{\mathsf{T}}\,\mathrm{d}\mathbf{k}, \tag{21}$$

$$\nabla_w L = (D_w\mathbf{F}_\theta)^{\mathsf{T}}\mathbf{v} + \mathsf{B}_w^{\mathsf{T}}(I-\mathsf{A}^{\mathsf{T}})^{-1}\boldsymbol{\eta}. \tag{22}$$

The direct-generation term has no ϕ component. Taking blocks recovers Eqs. (10)–(12). The inverse is a finite sum because the recurrence is causal, and reverse-mode differentiation evaluates these paths without constructing A.

The cotangent λ_i is the gradient with respect to the complete KV output of writer i, across all layers. Within one writer, an early-layer page may also influence a later-layer page. That intra-writer dependency belongs inside the local Jacobian and is not a separate temporal edge. The accumulated `.grad` of a raw intermediate tensor can include such internal use and is not the external λ_i.

A.2. Proof of Proposition 1

Proof. Fix the recorded inputs $\mathcal{D} = \{\bar{z}_i, \bar{x}_i^{\mathrm{ctx}}\}_{i=0}^{n}$ of one update and let $L_{\mathcal{D}}(\theta,\phi)$ denote $L_{\mathrm{DMD}}(\tilde{X})$ of Sec. 3.1 with Eqs. (1) and (2) evaluated on them. We compare Algorithm 1 with the uncheckpointed recurrence of Sec. 3.1 on $\nabla_{\theta,\phi}L_{\mathcal{D}}(\theta,\phi)$: the recorded latents are constants, and every dependency through the historical writes KV_j is retained.

At the segment level the differentiable computation consists of the exit prediction $\tilde{x}_i^{t_s}$ of Eq. (1) and the write KV_i of Eq. (2), and both depend on earlier segments only through their writes KV_j, $j < i$. The segment computations therefore form a directed acyclic graph ordered by time, with the terminal write KV_n a formal node that Algorithm 1 omits.

Both executions start from an empty history. Suppose their writes agree through segment $i-1$. Segment i then receives the same recorded inputs, historical KV, visibility and positions in both executions, so its exit prediction and its write agree. By induction all segment outputs are identical.

Differentiating the state extension of Eq. (8) gives Eq. (9). Its identity block preserves the derivatives of earlier writes, and its lower block carries the dependence of the current write on the parameters and on the historical KV. Reversing these dependencies accumulates gradient from later generation and writer nodes, giving Eq. (10) with the finite causal expansion of Appendix A.1. Summing the local parameter contributions gives Eqs. (11)–(12) without double-counting temporal paths.

Committed pages preserve the historical prefix seen by each segment (Eq. (8)). In backward, checkpoint replay uses the same prefix, group entry state, parameters and randomness as the original forward (Eq. (16)), so it reconstructs the same local computation and the same local Jacobian. Replacing the retained internal activations by this reconstruction leaves every vector–Jacobian product unchanged, and the state extension runs outside the reconstructed map, so replay cannot alter the history. Reverse-mode differentiation therefore produces the same parameter gradients as the uncheckpointed recurrence, in exact arithmetic. □

The proposition is an exact-arithmetic statement: attention kernels, accumulation orders and nondeterministic operations change floating-point results without changing the abstract graph. Appendix B.3 reports the numerical agreement of the implementation.

A.3. Two distinct temporal routes

Expanding the writer differential gives

$$\mathrm{dKV}_i = D_\theta \mathrm{KV}_i \, \mathrm{d}\theta + D_\phi \mathrm{KV}_i \, \mathrm{d}\phi + \sum_{j<i} D_{\mathrm{KV}_j} \mathrm{KV}_i \, \mathrm{dKV}_j. \tag{23}$$

The first two terms describe how parameters directly change a write with its current history fixed. The last term propagates a change in earlier memory into later memory. Combining this writer differential with Eq. (6) gives two temporal routes from a past write to the loss: direct-use paths

$$\mathrm{KV}_j \longrightarrow \tilde{x}_i^{t_s} \longrightarrow L_{\mathrm{DMD}}$$

and recurrent-write paths

$$\mathrm{KV}_j \longrightarrow \mathrm{KV}_r \longrightarrow \tilde{x}_i^{t_s} \longrightarrow L_{\mathrm{DMD}}, \qquad j < r < i.$$

The sampled context $\bar{x}_r^{\mathrm{ctx}}$ is detached (Eq. (2)), so neither route passes through the sampled prediction.

Since the loss has no direct dependence on the written KV, detaching the historical pages from every generation call removes all gradient to writer-only parameters, even when the dependencies between successive writes are retained (Eqs. (10) and (12)).

A.4. Role-specificity of the writer residual

At fixed θ and a history supplied as an independent tensor,

$$D_\phi G_\theta(\bar{z}_i, t_s, \mathrm{KV}_{<i}) = 0. \tag{24}$$

The total derivative through a writer-produced history is

$$\frac{\mathrm{d}\tilde{x}_i^{t_s}}{\mathrm{d}\phi} = \sum_{j<i} D_{\mathrm{KV}_j} \tilde{x}_i^{t_s} \frac{\mathrm{dKV}_j}{\mathrm{d}\phi}. \tag{25}$$

Generation therefore depends on the residual only through the historical KV it produces.

A.5. Gradients through the mutable cache

The matched computation. The mutable cache and the page table compute the same attention: for each segment, layer and role, attention reads the K/V written by the earlier segments together with the current segment's own K/V, and only the writer outputs become historical state. The two implementations differ in how these values are stored and connected to autograd. The matched comparison keeps the writer's sampled input detached as in Eq. (4) and enables the gradient through the historical K/V.

The distinction between state and its computation. A future loss needs the local derivatives of earlier writers, not only their KV values. Without rematerialization or offloading, the corresponding saved activations survive until backward. If $A_i^{\mathrm{gen}}, A_i^{\mathrm{mem}}$ are the internal saved-tensor footprints of the two roles, this includes a contribution proportional to $\sum_i (A_i^{\mathrm{gen}} + A_i^{\mathrm{mem}})$, in addition to historical state. Opening the gradient thus creates a temporal activation-lifetime problem, which the page execution resolves with local rematerialization.

A reproducible historical boundary. Rematerialization requires the same effective inputs as the original computation [18]. If an invocation relies on a mutable buffer C that is later overwritten or shifted, reading its final contents need not recover its original history:

$$\mathrm{Read}(C_{\mathrm{final}}, i) \neq \mathcal{H}_i \quad \text{in general.} \tag{26}$$

An overwritten saved tensor can raise a version-counter error, and rereading changed external state during checkpointing can reconstruct a different function. SF already includes block checkpointing, and the issue is preserving a valid historical boundary after memory gradients are opened. The persistent pages of Sec. 3.2 meet both requirements.

B. Implementation and experimental details

B.1. Experimental setup

Training uses four denoising steps at timesteps $1000/750/500/250$ with shift 5, on 16 latent channels at 60×104. Each generator update follows five critic updates. AdamW uses betas $(0, 0.999)$ and weight decay 0.01, the backbone learning rate is 2×10^{-6}, the writer-residual learning rate 10^{-4} and the EMA decay 0.99.

Inference keeps a sink of 4 frames, a FIFO window of 16 and the current frame in the frame-wise setting, and a $3 + 15 + 3$ layout in the chunk-wise setting, with the writer residual active only at cache refresh.

VBench at 5 s [8] evaluates 944 prompts with 5 videos each plus 75 flicker prompts with 25 videos each, and reports the Total, Quality and Semantic scores. VBench-Long

at 60 s uses 326 prompts and 241 latent frames (240 chunk-wise), at 240 s 963 latent frames, and reports subject and background consistency, motion smoothness, dynamic degree, aesthetic and imaging quality, and flicker on the static-filtered subset.

Training seeds. Frame-wise runs start from the official CD initialization and train for 1500 steps at 4 GPUs $\times$ batch 2. Every configuration is trained with seeds 1, 10 and 20, the seeds of Tables 5 and 6. Beyond these, CF++, SGF and the writer residual add seed 2 and SGF adds seed 30, and these runs enter the all-seed aggregates of Table 12 and Tables 9–11. The controls use the seed counts of Table 12, and chunk-wise runs start from the chunk-wise CD initialization. SGF and the writer residual have a second run of seed 1, and these same-seed repeats measure training variability: at 60 s the two SGF runs of seed 1 differ by 3.4 points of aesthetic quality and 31.9 of dynamic degree, and the two writer-residual runs by 3.4 and 15.5. Repeats are averaged before the mean $\pm$ SD over training seeds is taken.

B.2. Implementation

Each layer stores the post-RoPE K and V of the committed frames in one contiguous `_PageTable` of shape $[B, (T-1)L, n, d]$, allocated on first use in the attention dtype, and a committed slot is written once. The stored values stay outside the autograd graph, and a `_PageHandle`, an expanded scalar placeholder, keeps the gradient connection to each writer. A `PageRead` snapshot records the table, the visible length and the handles, and later appends leave its prefix unchanged. Untracked evaluations use the next uncommitted slot as scratch. Differentiable reads concatenate the historical values with the current KV through `_PageCat`, which returns one contiguous gradient per page, so a small retained view does not pin the gradient of the whole concatenation (code in Appendix B.4).

Non-reentrant checkpoint groups hold five blocks, with the patch embedding in the first group and the prediction head in the last, and RNG preservation is off because the blocks draw no random numbers. Differentiable cross-attention recomputes the text KV inside replay, and untracked sampling shares a cached text KV, outside any checkpointed region, until `end_rollout`. Calls enter through the module's `__call__` hooks so that FSDP gathers and casts the parameters, and RoPE and attention-length metadata are reused. The writer omits its output head and the final block's unused attention output, and no final-frame write is committed because it has no reader.

Execution variants. The default execution reuses the gradient-bearing exit prediction as the detached context, which removes a duplicate exit evaluation. Per-rank exit choices still require matching FSDP collective call sequences, a scheduling detail that Algorithm 1 omits. Fused batches the noisy and write roles with separate within-frame visibility and a writer-only residual mask, and obtains the detached write input through an untracked exit evaluation. Hidden keeps attention-input h pages in the graph and rebuilds the historical writer KV inside each reader's checkpoint region, in bf16, while a detached KV table serves the untracked reads. Fused+hidden combines the two. All four compute the same gradient target.

Precision. The common mixed-precision setting applies bf16 FSDP to the generator and the score networks, and the networks round timesteps consistently (937.5 becomes 936 in bf16). A second configuration keeps the teacher's time input in fp32 and casts its latents and text to bf16, so that the two score networks see the same timestep.

B.3. Validation

All four executions are compared with the two-pass reconstruction in fp64, at full size in bf16, under eight-GPU FSDP and in fp64 FSDP. In fp64 at batch sizes 2 and 3, with and without the residual and at $T = 8$ and 21, the outputs differ from the reference by 2×10^{-15} and the gradients by at most 2.2×10^{-14} (4.6×10^{-14} for the fused and hidden-state variants), and over the whole check the relative gradient error of every execution stays below 4.1×10^{-12} (2.9×10^{-12} against the pre-optimization traversal). A deliberate mixing of samples within one layer's K page produces gradient errors of 0.5–1.8% and is detected. At initialization $U = 0$ gives $\partial \Delta W / \partial Z = 0$ while $\partial \Delta W / \partial U$ need not vanish, so the connectivity checks read the gradient of U.

At full size ($T = 21$, spatial size 60×104, batch 2, deterministic attention) in bf16, the default execution matches the reference bitwise in outputs and gradients, and the fused and hidden-state variants match the outputs and differ in gradients by 0.4–0.9%. In 42-step trainer runs at 8×1 and 4×2, the default execution matches the reference across ranks and in the final weights. The fused and hidden-state variants match the first generator loss, and the cosine between their parameter displacement from the common initialization after 42 steps and that of the reference is 0.43–0.73, where two duplicate runs of the same training kernel give 0.68 and 0.53. Full-state resume with the residual at 4×2 is bitwise reproducible in the tested continuation.

Three-frame chunks agree with the two-pass reference in fp64 to 3×10^{-15} (outputs) and 2.6×10^{-14} (gradients) and agree in the three-path check below. With chunk-wise support added, the frame-wise path is bitwise unchanged. Chunk-wise quality is reported in Sec. 5.5.

The ordinary inference cache holds $18 + 3 = 21$ frames, and long inference uses a 4/16/1 window with top-aligned

RoPE. With identical checkpoint, prompt and noise, fp64 checks compare every frame of ordinary inference, long inference and the training generation path with the last exit and no context noise. The three paths agree bitwise or differ at rounding scale, and the differences grow without abrupt jumps over the rollout. Evaluations run after these checks.

B.4. Page execution in PyTorch

The following PyTorch excerpts implement the execution of Figure 2. They keep the operations and identifiers of the training code and omit dtype handling, the execution variants and the distributed bookkeeping. The implementation runs through ordinary module calls under FSDP with three small autograd functions and no custom kernel.

The pages exist only in the training rollout. Inference runs the ordinary cache path listed at the end of this appendix, where the same writer residual is active during the cache refresh and denoising uses the shared backbone without it. A page stores one segment's post-RoPE K and V at one layer, written once and connected to its writer through a data-free handle. When the inference cache covers the full training clip, the training and inference paths attend to the same historical K/V.

The traversal. The rollout follows the sampling of inference. Per segment, two forwards carry a graph: the exit forward G_θ, whose prediction enters the loss, and the write $\mathcal{K}_{\theta,\phi}$, whose output becomes the segment's pages. Every other sampling step runs without gradient.

```
store = PersistentKVStore(num_layers, max_frames=T - 1)
outputs = []
for i in range(T):
    hist = store.snapshot_all_layers()  # the prefix
    x_t = noise[:, i:i + 1]
    for s, t in enumerate(steps):  # sampling steps
        if s == exit_step:  # exit forward, with graph
            out = noisy(x_t, t, hist)
            outputs.append(out)
            x0 = x0_exit = out.detach()  # stop-gradient
        else:
            with torch.no_grad():
                x0 = noisy(x_t, t, hist)
        if s < len(steps) - 1:
            x_t = add_noise(x0, steps[s + 1])
    with torch.no_grad():  # the write input
        x_ctx = add_noise(x0_exit, context_t)
    if i < T - 1:  # the last segment writes no page
        pages = write(x_ctx, context_t, hist)
        store.append_all_layers(pages)  # commit once
store.end_rollout()
loss = video_loss(torch.cat(outputs, dim=1))
```

The one `detach` is the stop-gradient of the figure: the write reads the exit prediction as data, so the gradient reaches the writer only through its pages, never through its input. `end_rollout` drops only the prompt's cross-attention K/V that the forwards without gradient shared, and the pages stay, referenced by the graph, until the backward pass is done.

Write-once slots and the commit. Each layer allocates one contiguous page table on first use, at its full size. The page of segment i occupies rows $[iL, (i+1)L)$, and the commit writes that slot once, under `no_grad` and outside any checkpoint region.

```
class _PageTable:
    def put(self, slot, k, v):
        L = k.shape[1]
        if self.k is None:  # allocated once, appended to
            shape = (k.shape[0], self.max_frames * L,
                     *k.shape[2:])
            self.k = k.new_empty(shape)
            self.v = v.new_empty(shape)
        rows = slice(slot * L, (slot + 1) * L)
        self.k[:, rows].copy_(k)
        self.v[:, rows].copy_(v)
        return self.k[:, rows], self.v[:, rows]

class PersistentKVStore:
    def append_all_layers(self, pages):
        # pages = (k0, v0, k1, v1, ...), the writer output
        for l in range(self.num_layers):
            k, v = pages[2 * l], pages[2 * l + 1]
            kh = self.k_handles[l]
            vh = self.v_handles[l]
            with torch.no_grad():
                # the commit: outside any graph
                self.tables[l].put(self.length, k, v)
            if k.requires_grad:
                # the page's place in the graph
                kh.append(_PageHandle.apply(k))
                vh.append(_PageHandle.apply(v))
        self.length += 1

    def snapshot_all_layers(self):
        # a later append cannot change a snapshot
        return tuple(
            PageRead(self.tables[l], self.length,
                     tuple(self.k_handles[l]),
                     tuple(self.v_handles[l]))
            for l in range(self.num_layers))
```

A committed slot is never written again, so a checkpointed forward recomputed in backward reads the history it saw in forward and has no write to re-execute.

Handles. The K/V data live in the page table, and the handles keep their autograd connections. `_PageHandle` takes the writer's output and returns a zero scalar expanded to the page's shape, so the graph records the writer's output as a parent without a second K/V-sized tensor. Its backward passes the incoming gradient unchanged to the `k`, `v` tensors the writer returned, and from there into the writer's own graph. A snapshot is a tuple of the table, its visible length and the handles of the pages that carry a graph, and later appends leave it unchanged.

```
class _PageHandle(torch.autograd.Function):
    @staticmethod
    def forward(ctx, page):  # the page's shape, no data
        return page.new_zeros(()).expand(page.shape)

    @staticmethod
    def backward(ctx, grad):  # the gradient, unchanged
        return grad

class PageRead(NamedTuple):  # one layer's history
    table: _PageTable
    length: int
    k_handles: tuple
    v_handles: tuple
```

Reading through handles. A forward with gradient attends to the history slots plus its own K/V. `_PageCat` concatenates a view of the visible slots with the segment's own K/V into a tensor that lives only inside the rematerialized region. Its backward splits the history's gradient page by

page and returns each slice through the corresponding handle. Every returned slice, the segment's own included, is cloned into its own contiguous buffer, so a retained page gradient does not retain the gradient of the whole concatenation. A forward without gradient, every sampling step and every forward of the critic step, writes its own K/V into the next free slot as scratch and attends to a view of the table, so flash attention receives the same bytes either way.

```
class _PageCat(torch.autograd.Function):
    @staticmethod
    def forward(ctx, history, own, *handles):
        ctx.sizes = [h.shape[1] for h in handles]
        ctx.split = history.shape[1]
        return torch.cat([history, own], dim=1)

    @staticmethod
    def backward(ctx, grad):
        def owned(g):  # one contiguous buffer per page
            fmt = torch.contiguous_format
            return g.clone(memory_format=fmt)
        pages = grad[:, :ctx.split].split(ctx.sizes, 1)
        return (None, owned(grad[:, ctx.split:]),
                *(owned(g) for g in pages))

# the self-attention of one block
k, v = self.k(x), self.v(x)
if self.memory_lora and write_page:  # phi: write only
    k = k + self.k_memory_lora(x)
    v = v + self.v_memory_lora(x)
k = rope(self.norm_k(k))
table, n = history.table, history.length
if not torch.is_grad_enabled():  # sampling: in place
    k, v = table.put(n, k, v)  # scratch slot
    k_attn = table.k[:, :(n + 1) * L]
    v_attn = table.v[:, :(n + 1) * L]
else:  # with graph: read via handles
    k_attn = _PageCat.apply(table.k[:, :n * L], k,
                            *history.k_handles)
    v_attn = _PageCat.apply(table.v[:, :n * L], v,
                            *history.v_handles)
y = self.o(flash_attention(q, k_attn, v_attn))
return (y, k, v) if write_page else y
```

The writer residual (ϕ). The `write_page` flag adds the writer residual to the K and V projections of every block. The exit forward and all sampling steps use the backbone projections alone. The up-projection is zero-initialized, so at step 0 the written KV equals the pretrained one.

```
class MemoryLoRA(nn.Module):  # dW h = up(down(h))
    def __init__(self, dim, rank=64, alpha=None):
        self.scale = (alpha or rank) / rank
        self.down = nn.Linear(dim, rank, bias=False)
        self.up = nn.Linear(rank, dim, bias=False)
        nn.init.normal_(self.down.weight, std=1 / rank)
        nn.init.zeros_(self.up.weight)  # dW = 0 at init

    def forward(self, x):
        return self.up(self.down(x)) * self.scale
```

Checkpoint groups. Both graph-carrying forwards run their blocks in non-reentrant checkpoint regions of g blocks ($g = 5$ in our runs, six regions over 30 blocks). A region is handed only the histories of its own blocks, so a snapshot's handles enter the graph where they are read. Only the region inputs are kept between forward and backward, and the pages, being region outputs, are always kept. Nothing in a block draws random numbers, so no RNG state is saved.

```
def _persistent_blocks(self, x, histories, write_page):
    n = len(self.blocks)
    g = self.persistent_checkpoint_group
    pages = []
    for first in range(0, n, g):
        last = min(first + g, n)
        x, *page = torch.utils.checkpoint.checkpoint(
            self._run_blocks, first, last, x,
            histories[first:last], write_page,
            use_reentrant=False, preserve_rng_state=False)
        pages += page
    return x, pages
```

`_run_blocks` runs blocks `first` .. `last`-1 against their histories and returns the residual stream plus, for a write, those blocks' pages; the first region also runs the patch embedding and the last region of an exit forward runs the output head, so neither the embedded frame nor the head input is retained.

The two forwards. The writer has no head, and its last block computes only its page, since nothing reads that block's output.

```
def forward_noisy_frame(self, latent, histories):  # G
    x, _ = self._persistent_blocks(
        latent, histories, write_page=False)
    return self.unpatchify(x)  # the prediction

def forward_clean_writer(self, latent, histories):  # K
    _, pages = self._persistent_blocks(
        latent, histories, write_page=True)
    return tuple(pages)  # (k0, v0, k1, v1, ...), no head
```

Backward. No code is specific to the backward pass. Autograd carries the gradient of each read through `_PageCat` and the handles to the `k`, `v` the writer returned and into the writer's own regions, which are recomputed in turn, and the same route continues from a writer to the earlier pages it read. Replay never writes: the committed slots are read-only, and the tables are released when the last page gradient has been consumed.

The mutable cache. The ordinary path keeps one K/V buffer per layer and writes each segment into it in place, inside the block's checkpoint region:

```
# the ordinary path: a block inside its checkpoint region
for i, block in enumerate(self.blocks):
    x = checkpoint(block, x, kv_cache=kv_cache[i], ...)

# inside the block's attention
k, v = self.k(x), self.v(x)
kv_cache["k"][:, lo:hi] = k  # in place, with graph
kv_cache["v"][:, lo:hi] = v
k_attn = kv_cache["k"][:, :hi]
v_attn = kv_cache["v"][:, :hi]
y = self.o(flash_attention(q, k_attn, v_attn))
```

With gradient enabled, the assignment makes the buffer a node of the segment's graph, and every later reader saves that buffer for its backward. When a region is recomputed in backward, the assignment runs again on the same tensor and attaches a second graph to it, while the readers' saved buffers and the recomputed graphs accumulate. On 5 s clips the first backward pass runs out of memory (Sec. 3.4). Self Forcing and Causal Forcing++ avoid this by cutting the cache after every segment, which is exactly the gradient dKV Forcing restores:

```
# Self Forcing and CF++: cut the cache after each segment
for entry in kv_cache:
    entry["k"] = entry["k"].detach()
    entry["v"] = entry["v"].detach()
```

Table 7. Execution variants of the single traversal at 8 GPUs × batch 1 (no residual, bf16): a full training cycle of five critic updates and one generator update; times are steady-state medians, peaks the maximum allocated memory across ranks during a generator update. The variants are defined in Appendix B.2, and all rematerialize in groups of five blocks. Bold and underline: best and second best per column.

method	cycle (s)	peak (GiB)
Two-pass reconstruction	78.2	36.1
default	66.9	36.1
fused	**59.5**	36.6
hidden	69.4	**31.8**
fused+hidden	<u>62.6</u>	<u>33.3</u>

The two-pass reconstruction. SGF keeps the mutable cache and the cut and recovers the gradient with a second forward. Pass 1 is the rollout without gradient: it builds the cache and records each segment's noisy input at the exit step and its write input. Pass 2 is one teacher-forced forward over the whole clip with gradient, in which the recorded write inputs provide the context KV and the recorded noisy inputs the predictions:

```
# pass 1: the rollout without gradient builds the cache
with torch.no_grad():
    for i in range(T):
        for s, t in enumerate(steps):
            if s == exit_step:
                noisy_at_t[:, i] = x_t  # kept for pass 2
            x0 = model(x_t, t, kv_cache)
            ...  # sampling as before
        x_ctx = add_noise(x0_exit, context_t)
        x_ctx_hat[:, i] = x_ctx  # kept for pass 2
        model(x_ctx, context_t, kv_cache)  # the write
# pass 2: one teacher-forced forward over the whole clip
output = model(noisy_at_t, t_exit,
               clean_x=x_ctx_hat.detach(), aug_t=context_t)
loss = video_loss(output)
```

Pass 2 recomputes every exit forward and every write, so the graph of the whole clip exists at once and is held until backward. The gradient equals the single-traversal one (Proposition 1), at the cost of the second pass and of holding the whole-sequence graph (Table 1).

B.5. Performance measurements

Table 1 gives the training cycles of the execution variants at 4 GPUs × batch 2. At 8 GPUs × batch 1, Table 7 summarizes them and Table 8 gives the complete statistics, with and without the residual.

Measurement definitions. The distributed harness runs the actual DMD trainer with CUDA synchronization on every rank before phase timestamps. A measured cycle sums five consecutive rank-0 step walls and contains five critic updates and one generator update. It includes text encoding, rollout, scoring, backward, FSDP communication, clipping, optimizer updates, ordinary trainer logging and data loading. Steady cycles omit startup, periodic empty-cache calls and the recovery window through the next generator update, and the final step. Generator-substep and phase medians are separate statistics, so their sum approximates a median cycle. Allocated peaks are measured with PyTorch counters reset at each region, then maximized over ranks and the stated steps, and they include resident tensors as well as activations. Reserved peaks measure the allocator pool, and neither counter includes all external CUDA allocations.

Execution variants. Table 7 summarizes the four executions of Table 1 at 8 GPUs × batch 1 by cycle time and generator peak, and Table 8 breaks that measurement down, with and without the residual. At 8×1, the default execution takes 66.9 s per cycle against 78.2 s for the two-pass reconstruction. The unoptimized traversal took 88.8 s: its generator update (29.86 s against 20.14 s for the reconstruction) repeated the exit prediction and paid page assembly and indexing overhead, which the optimizations below remove.

Default. The default execution reuses the exit evaluation, stores history continuously, reuses positional and length metadata, omits the unconsumed terminal write, and includes input and output computations in replay regions. Its critic step drops to 8.00 s while its generator update stays at 25.69 s, and five critic updates per cycle amplify the rollout saving. The measured two-pass code runs an unconditional reconstruction pass even in the no-gradient critic update (about 1.2 s) and spends about 72 ms per cached frame forward against about 50 ms in the page path.

Fused. The fused write executes the noisy and write roles together. At 8×1, the rollout plus gradient-forward phase falls from 11.09 to 6.91 s and backward from 7.56 to 4.72 s, and the generator update from 25.69 to 19.20 s. Critic time remains approximately 8 s. Batching reorganizes the two roles without changing their arithmetic, and the gain comes from higher device utilization and amortized execution overhead. Simultaneous intermediates raise the allocated peak slightly (Table 8).

Hidden and fused+hidden. Hidden-state pages retain attention inputs h instead of both K and V and reconstruct the latter when needed. Relative to the default execution, the generator peak falls (Table 8) while the generator update rises from 25.69 to 29.19 s and backward from 7.56 to 9.53 s, with the critic time unchanged. Fused+hidden combines this storage reduction with batching, with a generator update of 22.09 s. The effects are confined to the differentiable-memory path, and the no-gradient rollout is unchanged.

GPU layout. At 4×2 (Table 1), the large batching gain seen at 8×1 disappears: real batch 2 already improves utilization, role batching increases the internal batch to 4 and raises intermediate storage, and a non-contiguous historical-table slice causes an extra full-table copy in the

Table 8. Training cycles at 8 GPUs × batch 1 (bf16). Steady-cycle medians ± half the min–max range over six cycles. Memory in GiB (no-LoRA maximum over the eight ranks, resident memory at critic-rollout entry). Bold and underline: best and second best per column.

Method	Cycle, no LoRA (s)	vs. two-pass	Cycle, LoRA (s)	vs. two-pass	Gen. peak	Critic peak	Resident	Step-0 peak	All-step peak	Reserved
Two-pass	78.2 ± 1.1	+0.0%	81.0 ± 1.0	+0.0%	36.1	29.3	20.4	59.0	59.0	60.8
Unoptimized	88.8 ± 1.2	+13.6%	96.3 ± 1.9	+18.9%	36.4	23.5	14.7	20.2	36.4	58.7
default	66.9 ± 0.9	−14.5%	67.9 ± 2.8	−16.2%	36.1	23.5	14.7	20.2	36.1	58.3
fused	**59.5** ± 1.2	−23.9%	**61.8** ± 0.5	−23.7%	36.6	23.5	14.7	20.2	36.6	60.6
hidden	69.4 ± 1.9	−11.2%	75.2 ± 1.0	−7.2%	**31.8**	23.6	14.7	20.2	**31.8**	**56.2**
fused+hidden	<u>62.6</u> ± 0.6	−20.0%	<u>65.7</u> ± 0.7	−19.0%	<u>33.3</u>	23.6	14.7	20.2	<u>33.3</u>	<u>57.9</u>

no-gradient path at batch sizes above one. Fewer FSDP shards and twice as many local samples increase per-GPU state and activations. For the default execution, four-GPU wall time is longer but GPU-seconds per cycle decrease from $8(66.9) = 535.2$ to $4(97.3) = 389.2$, about 27%.

Checkpoint group size and production steps. Generator-only measurements of the checkpoint group size (weights excluded) give 20.5 GiB and 25.3 s for groups of one block, 17.3 GiB and 15.9 s for groups of five and 25.3 GiB and 13.7 s for groups of thirty, which sets the default of five. In production training at 4×2 with checkpoint saving, the default execution with the residual takes 21.0 s per step at a rank-0 allocated peak of 62.5 GiB, the fused execution 20.1 s at 65.9 GiB, and Self Forcing 21.2 s at 56.1 GiB.

Cold start and allocator state. The two-pass reconstruction's 59.0 GiB first-step peak at 8×1 comes from the first BlockMask construction and is distinct from its 36.1 GiB steady generator peak. Each periodic empty-cache flush costs 33–46 s including recovery, about 9–13 s per cycle. The initial flush is kept, since omitting it leaves the two-pass reconstruction with a 75.8 GiB allocator pool and no free device memory for NCCL.

Component accounting of one generator update. Figure 8 gives the measured memory composition of one generator update on rank 0 (4 H100 × batch 2, bf16, FSDP, 21 frames). The CUDA allocator trace is classified by allocation site and replayed on a 50 ms grid, the resident baseline (weights, optimizer states, EMA, and for Self Forcing and the two-pass reconstruction the cache kept between steps) is taken from the model and optimizer objects at step entry, and the phase durations are the steady-state medians of Table 1. The naive mutable cache uses the instrumented run itself and ends at its out-of-memory failure in the first backward. Dashed lines are the measured `max_memory_allocated` peaks.

Memory retention under checkpoint replay. The examined mutable implementation writes projected KV into an external buffer inside checkpointed forward. Non-reentrant backward replay repeats those writes and attaches new graph connections (including `CopySlices`) to externally retained tensors. Local replay graphs can consequently outlive their use. The saved-input closure also retains a nested cache reference. Disabling checkpointing removes replay but retains all forward activations, producing a different OOM mechanism.

The full-size generator-only run uses eight H100 GPUs, 30 layers, 21 frames, 1,560 tokens per frame and bf16/FSDP, without teacher, critic and optimizer. Original mutable-cache checkpointing ends forward at 15.36 GiB, peaks at 57.22 GiB and leaves 42.29 GiB after backward. Isolating only the replay-time cache graph writes leaves the forward footprint unchanged and lowers the peak to 28.71 GiB. Combining isolation with reference cleanup gives three stable rounds at 28.71 GiB, and pages with per-layer checkpointing peak at 28.09 GiB. This locates the main excess in backward lifetime.

Detaching the external cache alone or removing its saved-input reference alone leaves 42.29 GiB, and doing both reduces allocated memory to 7.13 GiB. In the full-DMD run, the mutable cache with checkpointing peaks at 69.15 GiB on the first generator update and runs out of memory on the second backward, without checkpointing it fails before frame 6 at 75.15 GiB, whereas pages with groups of five complete the same updates at 35.17 and 36.50 GiB. In separate four-frame, six-layer, width-128 tests across eight ranks, replay isolation changes no parameter gradient (maximum absolute difference zero), and a diagnostic repaired cache completes two full-DMD updates.

C. Additional experiments and results

C.1. Controlling the memory gradient and writer parameters

Gradient scaling is a backward intervention. Gradient scaling and rollout dropout are the two controls with which Sec. 5.6 moves the model along the fidelity–motion tradeoff (Table 12): scaling sets the strength of the memory gradient at every writer-output interface, and rollout dropout sets how often the residual takes part in a training rollout. The scale α is the code's `memory_grad_scale`. Each

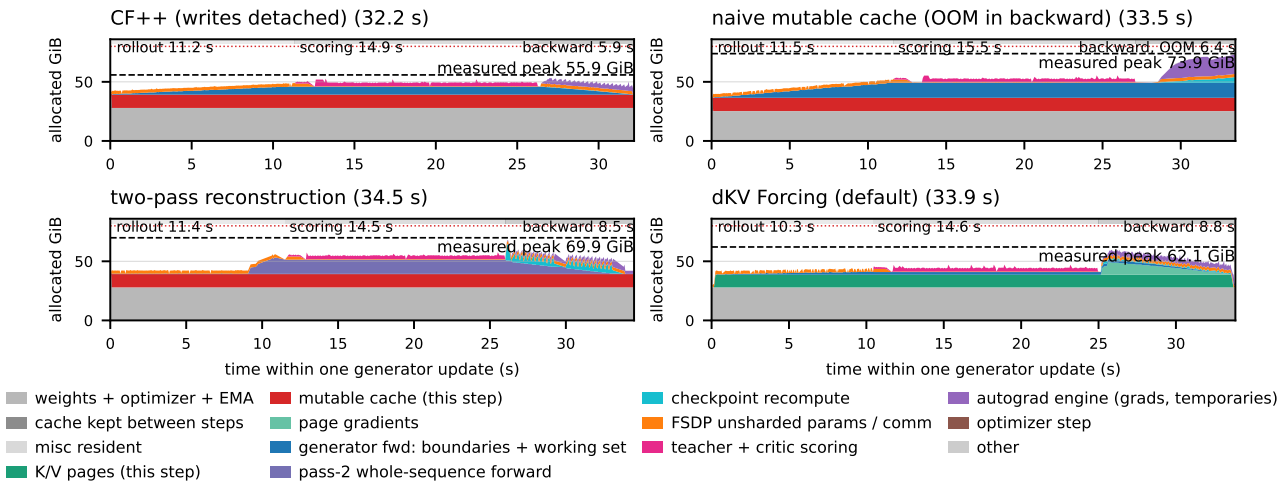


Figure 8. Measured component accounting of one generator update for the four training schemes. Panel titles give the generator-update time. CF++ (detached writes) keeps only exit-forward boundaries and its backward is short. The naive mutable cache keeps both forwards' boundaries and the sampled trajectory, and its backward exhausts memory while replay attaches new graph to the external cache (purple: engine-allocated gradients and temporaries). The two-pass reconstruction keeps no graph in the rollout but retains the whole second-pass forward (violet) and replays it layer by layer in backward (cyan), with the cache resident between steps (red). dKV Forcing allocates the page table once (green), keeps only group entry states during the rollout, and in backward adds one contiguous gradient per page (light green) and the local replay of five-block groups.

writer output passes through an identity-forward operator whose backward multiplies by α. For the joint output-interface abstraction used in Eq. (10), the resulting writer cotangent satisfies

$$\boldsymbol{\lambda}^{(\alpha)} = \alpha(\boldsymbol{\eta} + \mathsf{A}^{\mathsf{T}}\boldsymbol{\lambda}^{(\alpha)}) = \sum_{q=0}^{n} \alpha^{q+1}(\mathsf{A}^{\mathsf{T}})^{q}\boldsymbol{\eta}. \tag{27}$$

Thus a route crossing h scaled page interfaces receives α^h. At $\alpha = 1$ this is the exact gradient, and intermediate values alter the backward rule. In the step-1,400 measurements, cancellation between opposed one-hop and multi-hop terms makes the *net* memory norm at $\alpha = 0.5$ about 0.55–0.58 of its $\alpha = 1$ value.

Whole-rollout memory-gradient dropout chooses $\alpha = 0$ with probability p and 1 otherwise. At a fixed recorded computation its expected update is the mixture $pg^{(0)} + (1-p)g^{(1)}$, and the scaled gradient $g^{(1-p)}$ differs from it because recurrent paths cross multiple interfaces.

An independent writer. The independent-writer variant introduces a second 1.3B model initialized from the same CD checkpoint. With parameters ψ, replace Eq. (2) by $\mathrm{KV}_i = \mathcal{K}_\psi(\bar{x}_i^{\mathrm{ctx}}, \tau, \mathrm{KV}_{<i})$. The generator receives the direct-role parameter gradient, and ψ receives the memory gradient. The writer has its own FSDP wrapper, optimizer group, gradient clipping, EMA, checkpoint state, text cache and inference window configuration. At identical initialization, generator gradients differ from shared-model $\alpha = 0$ by 0.435% (repeat noise 0.434%), the writer gradient differs from the full-minus-direct term by 0.97%, and the summed gradient from the full gradient by 0.66%. A perturbed-writer inference check agrees across paths in fp64, and removing the writer changes outputs. Resume and short training checks passed. In quality, the dedicated writer of Table 4 reaches 64.9 ± 0.6 aesthetic and 71.6 ± 0.7 imaging points over three seeds with the highest mean subject consistency of the table (98.6 ± 0.4), at two generator-sized parameter sets and a full-network write forward, and SGF, the residual and the dedicated network form a monotone sequence in how much of the writing role is free to adapt and in aesthetic quality and subject consistency.

Real-DMD gradient measurement scope. At step 1,400, two batches use the next training batch and matched noise to compare scales 0/0.25/0.5/0.75/1, a detached writer-history path, and selected-page cuts. Repeat noise is 0.5–1%. The memory/direct norm ratio is 1.00–1.06 for the attenuated checkpoint and 1.04–1.06 for the full-gradient checkpoint. Their raw cosines are -0.10 to -0.12 and -0.08 to -0.14, respectively, and Adam-normalized cosines are -0.15 to -0.17 and about -0.22. LoRA has zero direct-role contribution. These same-sample decompositions are local diagnostics of the trained checkpoints.

Table 9. Every frame-wise configuration, VBench-Long standard prompts at 60 s, step 1500, mean ± SD over n training seeds (same-seed repeats averaged first).

configuration	n	aesthetic	dynamic	imaging	subject
CF++ (writes detached)	4	51.6 ± 2.4	74.9 ± 15.8	62.7 ± 1.8	94.8 ± 1.1
SGF ($\alpha = 1$)	5	57.2 ± 2.6	38.7 ± 16.3	68.0 ± 1.8	96.7 ± 0.8
writer LoRA, $\alpha = 0.5$	4	62.5 ± 2.7	33.9 ± 24.0	69.5 ± 1.1	98.1 ± 0.7
SGF, $\alpha = 0.25$	4	54.8 ± 1.8	48.6 ± 32.6	66.4 ± 1.5	96.0 ± 1.8
SGF, $\alpha = 0.5$	4	54.9 ± 2.5	46.1 ± 21.2	65.0 ± 1.3	96.0 ± 1.3
writer LoRA, $\alpha = 0.25$	4	58.6 ± 2.8	14.5 ± 9.1	66.9 ± 3.0	97.5 ± 0.6
writer LoRA, $\alpha = 0.75$	4	62.4 ± 2.6	30.8 ± 26.0	69.8 ± 0.6	98.1 ± 0.6
dedicated writer	3	64.9 ± 0.6	26.0 ± 18.2	71.6 ± 0.7	98.6 ± 0.4
writer LoRA ($\alpha = 1$)	4	62.1 ± 1.4	22.6 ± 14.4	69.6 ± 0.6	97.9 ± 0.6
writer LoRA + rollout dropout 0.5 (ours)	4	60.1 ± 1.2	74.0 ± 28.7	69.0 ± 1.1	96.6 ± 1.0
writer LoRA + rollout dropout 0.25	4	60.4 ± 0.8	38.8 ± 19.9	69.4 ± 0.4	97.6 ± 0.5
writer LoRA + rollout dropout 0.75	4	56.1 ± 4.0	75.2 ± 18.1	66.8 ± 2.6	95.7 ± 1.0

Table 10. Every chunk-wise configuration (three latent frames per segment, cache $3 + 15 + 3$), VBench-Long standard prompts at 60 s, step 1500, mean ± SD over n seeds.

configuration	n	aesthetic	dynamic	imaging	subject	background	smoothness	flicker
CF++ (writes detached)	3	57.8 ± 1.6	50.6 ± 15.8	65.6 ± 1.2	96.8 ± 0.7	95.7 ± 0.7	98.3 ± 0.5	98.8 ± 0.7
SGF ($\alpha = 1$)	3	58.1 ± 0.1	48.4 ± 19.8	67.8 ± 1.4	96.9 ± 0.5	95.7 ± 0.6	98.6 ± 0.5	98.8 ± 0.3
writer LoRA	3	59.6 ± 0.8	61.9 ± 19.7	68.7 ± 1.7	96.8 ± 0.8	95.8 ± 0.1	98.6 ± 0.4	99.0 ± 0.3

Table 11. Every configuration evaluated at 240 s (VBench-Long standard prompts, 963 latent frames), step 1500, mean ± SD over all available training seeds.

configuration	n	aesthetic	dynamic	imaging	subject	background	smoothness	flicker
CF++ (writes detached)	4	47.7 ± 3.2	77.8 ± 14.5	60.8 ± 3.0	93.9 ± 0.8	94.4 ± 0.6	97.8 ± 1.3	98.4 ± 0.3
SGF ($\alpha = 1$)	5	51.9 ± 2.3	38.8 ± 15.0	65.0 ± 2.6	95.9 ± 0.8	95.5 ± 0.6	98.6 ± 0.3	98.2 ± 0.7
writer LoRA ($\alpha = 1$)	4	58.9 ± 2.2	22.1 ± 16.0	67.6 ± 1.1	97.3 ± 0.8	96.3 ± 0.4	99.0 ± 0.2	99.1 ± 0.3
writer LoRA + rollout dropout 0.5 (ours)	4	57.4 ± 0.3	74.3 ± 28.0	68.2 ± 0.4	96.2 ± 0.8	94.9 ± 0.6	97.8 ± 0.8	98.6 ± 0.1 (n=3)
chunk-wise CF++	3	52.5 ± 4.3	50.4 ± 16.1	62.3 ± 2.9	95.9 ± 0.5	95.3 ± 0.4	98.0 ± 0.7	98.7 ± 0.6
chunk-wise SGF	3	53.2 ± 1.0	46.8 ± 20.4	65.4 ± 2.7	96.4 ± 0.5	95.4 ± 0.5	98.6 ± 0.5	98.7 ± 0.3
chunk-wise writer LoRA	3	55.7 ± 2.2	57.1 ± 24.3	66.2 ± 2.3	96.6 ± 0.6	95.4 ± 0.1	98.8 ± 0.3	98.9 ± 0.2

Table 12. Strength of the writing-path training versus 60 s fidelity and motion (VBench-Long standard prompts, frame-wise, mean ± SD over n seeds). α scales the backward gradient at each writer-output interface. Upper block: SGF with a scaled memory gradient. Middle block: writer LoRA. Lower block: writer LoRA ($\alpha = 1$) with a control that changes how the residual is trained. Per block, best value per column in bold, second best underlined.

	configuration	n	aesthetic	dynamic	imaging
no residual	$\alpha = 0$ (CF++, writes detached)	4	51.6 ± 2.4	$\mathbf{74.9} \pm 15.8$	62.7 ± 1.8
	$\alpha = 0.25$	4	54.8 ± 1.8	$\underline{48.6} \pm 32.6$	$\underline{66.4} \pm 1.5$
	$\alpha = 0.5$	4	$\underline{54.9} \pm 2.5$	46.1 ± 21.2	65.0 ± 1.3
	$\alpha = 1$ (SGF)	5	$\mathbf{57.2} \pm 2.6$	38.7 ± 16.3	$\mathbf{68.0} \pm 1.8$
writer LoRA	$\alpha = 0.25$	4	58.6 ± 2.8	14.5 ± 9.1	66.9 ± 3.0
	$\alpha = 0.5$	4	$\mathbf{62.5} \pm 2.7$	$\mathbf{33.9} \pm 24.0$	69.5 ± 1.1
	$\alpha = 0.75$	4	$\underline{62.4} \pm 2.6$	$\underline{30.8} \pm 26.0$	$\mathbf{69.8} \pm 0.6$
	$\alpha = 1$	4	62.1 ± 1.4	22.6 ± 14.4	$\underline{69.6} \pm 0.6$
dropout	rollout dropout $p = 0.25$	4	$\mathbf{60.4} \pm 0.8$	38.8 ± 19.9	$\mathbf{69.4} \pm 0.4$
	rollout dropout $p = 0.5$ (ours)	4	$\underline{60.1} \pm 1.2$	$\underline{74.0} \pm 28.7$	$\underline{69.0} \pm 1.1$
	rollout dropout $p = 0.75$	4	56.1 ± 4.0	$\mathbf{75.2} \pm 18.1$	66.8 ± 2.6

C.2. KV representation analyses and intervention controls

Table 2 compares generation and written KV across layers and checkpoints. The gap is already large at the untrained initialization and changes little after 1500 steps of training without the residual (paired difference K $+0.012 \pm 0.003$, V $+0.019 \pm 0.004$). A linear classifier separates the two representations at every layer, $\mathrm{KV}^{\mathrm{gen}}$ linearly encodes the noise level of its forward ($R^2 = 0.995$), and the attention-mass ratio reaches $2.5\times$ in layer 18.

Both representations encode the coarse content equally well (held-out R^2 of the clean latent 0.98 against 0.99), so the gap lies in the encoding with this content intact. The trained writer residual (writer LoRA, $\alpha = 1$) moves the write by 8–15% of this gap. Against $\phi = 0$ the relative difference is 0.034 for K and 0.054 for V, the token-wise cosine 0.999 and 0.998, and held-out linear separability 99% and 98%.

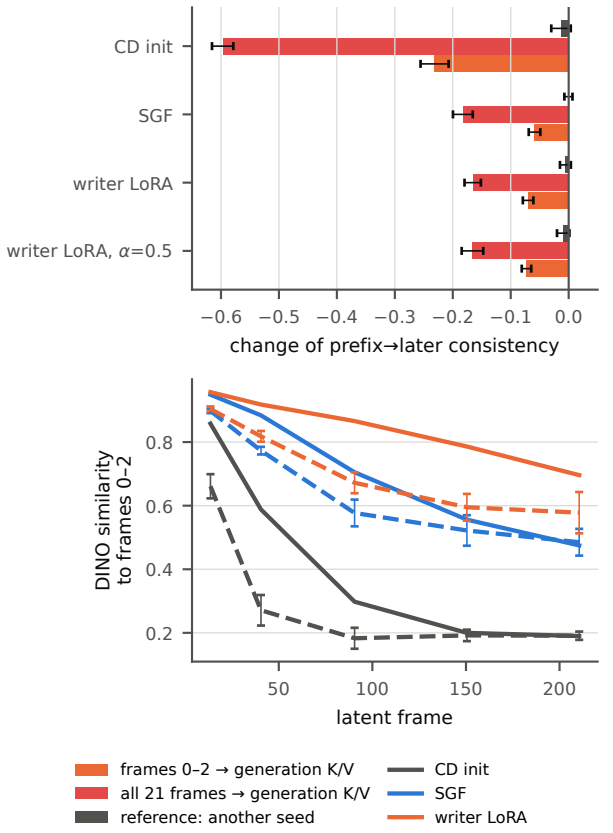


Figure 9. Cache intervention. Top: paired change of the prefix→later DINO consistency at 5 s when the memory of frames 0–2, or of all 21 frames, is replaced by generation K/V, with an independent seed as the reference scale (mean ± s.e.m. over 48 prompts). Bottom: 60 s rollouts (16 prompts), similarity to the opening by latent-frame window, unchanged (solid) versus memory of frames 0–2 replaced (dashed).

Intervention controls. The cache intervention of Sec. 5.3 replaces the memory of the first three frames by their generation K/V, by skipping their refresh forward, with checkpoint, prompt, initial noise, re-noising noise and denoising schedule held fixed (Table 3, Figure 9). The consistency drop is present in 44–46 of 48 prompts for frames 0–2 and in 47–48 of 48 when all frames are replaced, and it is monotone in the number of intervened frames, a second seed used as the reference changes consistency by 0.00 ± 0.01, and in 60 s rollouts the deficit persists through latent frames 61–120 (15–30 s) in all three checkpoints tested and through frames 121–180 for the writer-LoRA model. The same three-frame intervention at frames 11–13 has a smaller latent effect at every matched lag and no consistency effect. Switching off only the residual at the write ($\Delta W = 0$) is a smaller intervention (next-frame change 0.14–0.26) that lowers consistency in the models with the residual when applied to all frames.

C.3. Fidelity versus motion

In the frame-wise regime, stronger writing-path training traces a broad trade-off between long-horizon fidelity and motion (Table 12, Figure 7). Without the residual the memory-gradient scale α moves both means monotonically: from the detached write ($\alpha = 0$) through $\alpha = 0.25$ and 0.5 to the full SGF, aesthetic quality rises from 51.6 to 57.2 while dynamic degree falls from 75 to 39. With the residual the mean scores move further in the same directions. With the residual in place, $\alpha = 0.5$ and 0.75 leave both scores within seed variation of $\alpha = 1$, and $\alpha = 0.25$ sits at 58.6 ± 2.8 aesthetic and 14.5 ± 9.1 dynamic degree. The residual, trained at fifty times the backbone learning rate, reaches the low-motion solution with a quarter of the memory gradient. Gradient scaling and rollout dropout are different operations on the backward pass: a route that crosses h scaled page interfaces receives α^h (Eq. (27)), whereas dropout removes the residual from whole rollouts and leaves the remaining gradient unscaled.

Rollout dropout disables the writer residual for an entire rollout with probability p in both the critic and the generator step, and the mean scores move monotonically with p (Table 12): $p = 0.25$ keeps 60.4 points of aesthetic quality at a dynamic degree of 38.8, $p = 0.5$ restores the dynamic degree of the detached writer (74.0) at 60.1, and $p = 0.75$ reaches 75.2 at 56.1.

C.4. Complete results

Tables 9–11 list the configurations at the standard training budget with all available seeds: frame-wise at 60 s, chunk-wise at 60 s, and every 240 s evaluation. The extended-training controls are reported in Table 12. Conventions are those of Sec. 5.1: mean ± SD over training seeds, same-seed repeats averaged first, flicker on the static-filtered subset (missing where no clip passed the filter). These all-seed aggregates differ from the matched-seed subsets in Tables 5 and 6, and each table states its seed scope.

C.5. Additional qualitative comparisons

Figures 10–16 show frames of the same prompt generated by the four frame-wise configurations of Table 6 (CF++ with detached writes, SGF, the writer LoRA, and the writer LoRA with rollout dropout 0.5, i.e. dKV Forcing) at 0, 48, 96, 144, 192 and 238 s, on seven 240 s generations from the 128 MovieGen prompts (the 240 s prompt set of SGF [35]; 963 latent frames; EMA weights, sampling seed 0, sink 4 + FIFO 16 + current 1). Figures 17–26 show the same four configurations at 0, 12, 24, 36, 48 and 59 s on ten 60 s generations (241 latent frames, same sampling) from the same prompt set. Figure 1 shows dKV Forcing alone, and the same four generations are compared with the baselines in

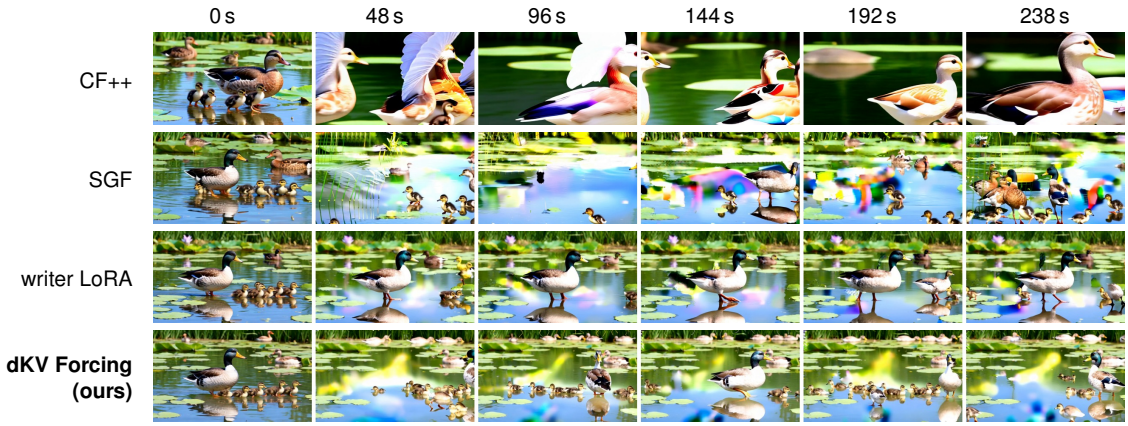


Figure 10. "Mother duck and six ducklings". Prompt: *A serene watercolor painting of a mother duck leading her six ducklings across a tranquil pond. The mother duck has a gentle expression, her feathers glistening in the sunlight, and she frequently glances back to ensure all her ducklings are safely following in a neat little line. The ducklings follow closely behind, their small heads bobbing up and down as they waddle along. The background features a peaceful pond with lily pads and ducks floating nearby, creating a harmonious and natural scene. A mid-shot from a slightly elevated angle captures the mother duck and her ducklings in motion.*



Figure 11. "Singer hitting a high note". Prompt: *A vibrant concert poster-style image of a singer performing on stage, their mouth wide open as they hit a high note. The singer, a young woman with striking blue eyes and long wavy blonde hair, is dressed in a black sequined dress with a plunging neckline and glittering accents. She stands confidently with one hand raised, exuding passion and energy. The background features a blurred stage with colorful lights and a grand piano in the corner, creating a lively and dynamic atmosphere. A medium shot from a slightly elevated angle capturing the singer's intense performance.*

Figure 4 (sword fight), Figure 18 (bubbles) and Figure 6 (cat and kitchen).

In these examples the detached-write model loses the scene within the first minute and degenerates into color fields or a different scene, SGF keeps a plausible scene but drifts to other subjects or layouts, and dKV Forcing keeps the subject and the layout through the full minute or the full four minutes while the subject keeps moving.

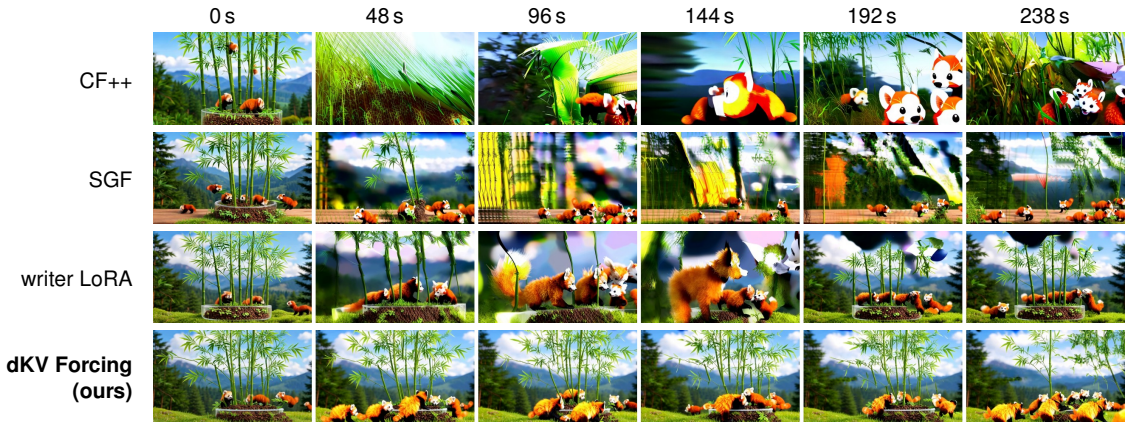


Figure 12. "Red pandas in a petri-dish bamboo forest". Prompt: *A high-resolution digital artwork in a realistic botanical style, showcasing a petri dish where a miniature bamboo forest thrives, complete with tiny red pandas running around. The bamboo stalks are slender and green, with delicate leaves swaying gently. The red pandas, with their distinctive reddish-brown fur and black legs, move playfully among the bamboo, sometimes climbing up the stalks or nibbling on leaves. The petri dish is filled with nutrient-rich soil, and the background is a blurred but recognizable forest landscape, with hints of distant mountains and clear blue skies. The entire scene exudes a sense of harmony and tranquility, capturing the wonder of nature in a microscopic world. A macro shot from a low angle, emphasizing the intricate details of the red pandas and the bamboo.*

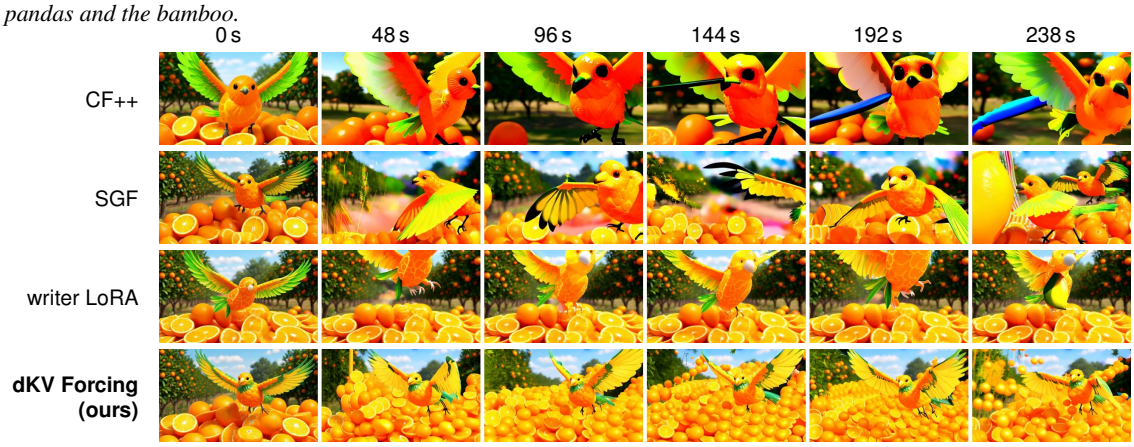


Figure 13. "Bird made of oranges". Prompt: *A dynamic and vibrant illustration in the style of a modern digital painting, depicting a bird crafted entirely from fresh oranges in vivid hues of orange and green. The bird is in flight, its wings spread wide and agile, as if rushing out of a large pile of freshly sliced oranges. The bird's body is composed of various sizes and shapes of oranges, with some segments showing the inner fruit flesh, adding a sense of freshness and vitality. Its eyes are small and black, giving it a lively and spirited expression. The background is a blurred orchard with rows of orange trees, some leaves fluttering in the wind. The scene captures the bird in mid-flight, with a slight tilt of its head and a spread of its wings, creating a sense of movement and energy. A close-up shot from a low angle, emphasizing the intricate details of the bird's form.*

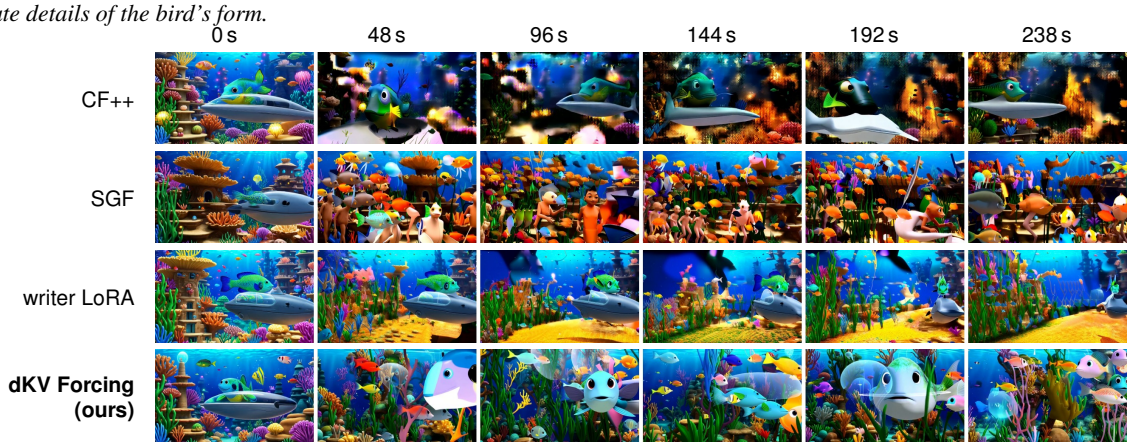


Figure 14. "Fish driving a submarine". Prompt: *A detailed CG illustration in a vibrant, realistic style, depicting a curious fish driving a small, sleek submarine. The fish has large, expressive eyes and vibrant scales in shades of blue and green, swimming gracefully inside the submarine. The submarine itself is compact and streamlined, with clear portholes allowing the fish to see the surroundings clearly. It navigates through an intricate underwater city filled with towering structures made of coral and shells, vibrant marine life swimming around, and colorful lights illuminating the scene. The background showcases a vivid, bioluminescent ocean with schools of fish and glowing jellyfish. A dynamic overhead view, capturing the fish's movement and the submarine's journey through the bustling underwater metropolis.*



Figure 15. “Man pushing a massive stone”. Prompt: *A dramatic action shot in the style of a superhero film, depicting a man pushing away a massive stone with superhuman strength. The man, muscular and determined, has short dark hair and a determined expression, his eyes fixed intently on the stone. He is wearing a tattered, loose-fitting shirt and shorts, emphasizing his powerful physique. His arms are strained as he pushes the stone, sweat glistening on his brow. The background shows rugged terrain with craggy rocks and a distant mountain range, adding to the epic feel of the scene. The stone is so large it covers most of the frame, creating a sense of scale. A dynamic, low-angle shot capturing the intense moment of exertion.*

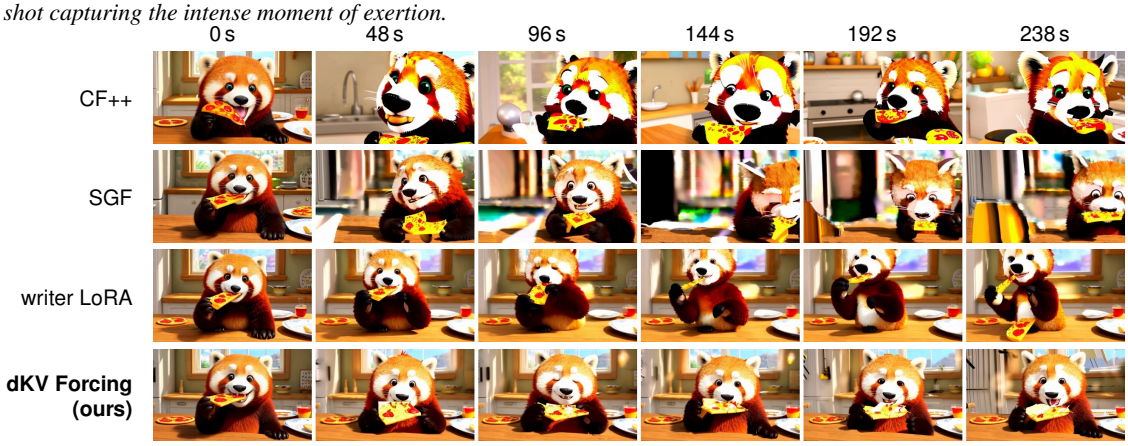


Figure 16. “Red panda eating pizza”. Prompt: *A vibrant and lively illustration in a whimsical cartoon style of a red panda taking a bite of a pepperoni pizza. The red panda has a playful expression, with large, round eyes and a mischievous grin. It has reddish-brown fur and black markings around its eyes and muzzle. The panda is sitting upright on a wooden table, one paw holding the pizza slice while it takes a big bite. The background features a cozy kitchen setting with a few scattered plates and utensils, and a window letting in warm sunlight. A close-up shot from a slightly lower angle, capturing the panda’s joyful moment.*

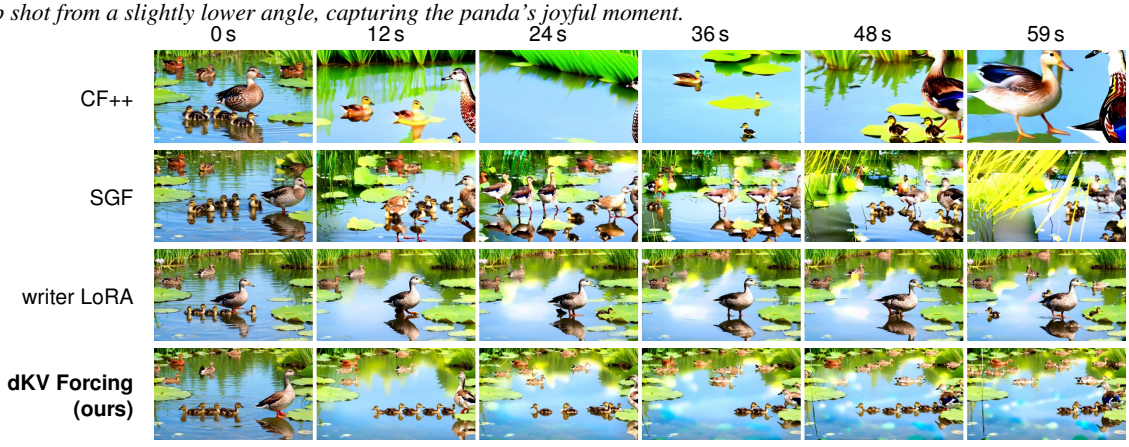


Figure 17. “Mother duck and six ducklings”. Prompt as in Figure 10.



Figure 18. "Child blowing bubbles". Prompt: *A children's illustration in a cheerful watercolor style, depicting a young child joyfully blowing bubbles in a sunny outdoor setting. The child has curly brown hair and a bright smile, standing with one hand extended to release a cluster of colorful soap bubbles. The bubbles float upward, creating a whimsical and playful scene. The background features a blurred garden with flowers, green grass, and a few trees, emphasizing the lively atmosphere. The child's eyes sparkle with delight, and their posture conveys excitement and energy. A close-up shot from a slightly lower angle, capturing the child's face and the bubbles in mid-flight.*

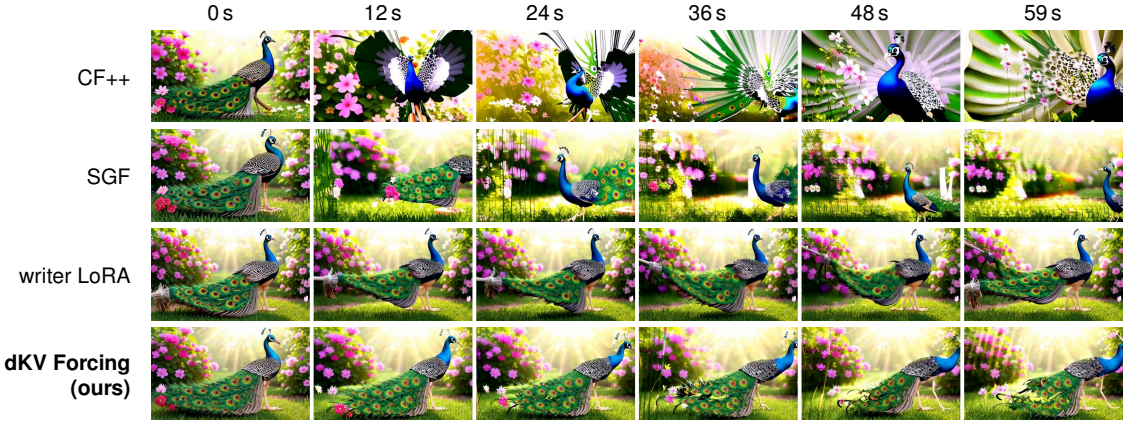


Figure 19. "Broom morphing into a peacock". Prompt: *A dynamic motion shot of a broom morphing surreally and magically into a peacock. The broom handle shortens and curves into a slender neck, while the bristles fan out into a magnificent tail. Vibrant colors and eye-shaped patterns emerge on the expanding feathers as the camera moves around, capturing every detail. A small head forms at the top, complete with a delicate crest. The transformation completes as the peacock proudly displays its newly formed plumage, standing tall and regal. The background features a soft, pastel garden with blooming flowers and gentle sunlight filtering through, enhancing the ethereal quality of the scene.*



Figure 20. "Rider in the Gobi Desert at sunset". Prompt: *A cinematic scene from a classic western movie, featuring a rugged man riding a powerful horse through the vast Gobi Desert at sunset. The man, dressed in a dusty cowboy hat and a worn leather jacket, reins tightly on the horse's neck as he gallops across the golden sands. The sun sets dramatically behind them, casting long shadows and warm hues across the landscape. The background is filled with rolling dunes and sparse, rocky outcrops, emphasizing the harsh beauty of the desert. A dynamic wide shot from a low angle, capturing both the man and the expansive desert vista.*

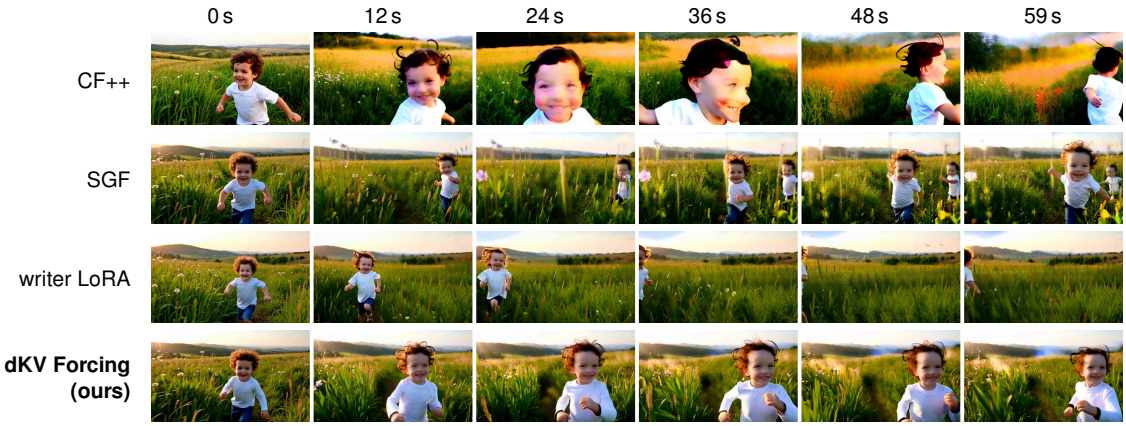


Figure 21. "Child running through tall grass". Prompt: *A handheld shot following a young child running through a field of tall grass, capturing the spontaneity and playfulness of their movements. The child has curly brown hair and a mischievous smile, arms swinging freely as they sprint across the green expanse. Their small feet kick up bits of grass and dirt, creating a trail behind them. The background features a blurred landscape with rolling hills and scattered wildflowers, bathed in warm sunlight. The photo has a natural, documentary-style quality, emphasizing the dynamic motion and joy of the moment. A dynamic handheld shot from a slightly elevated angle, following the child's energetic run.*

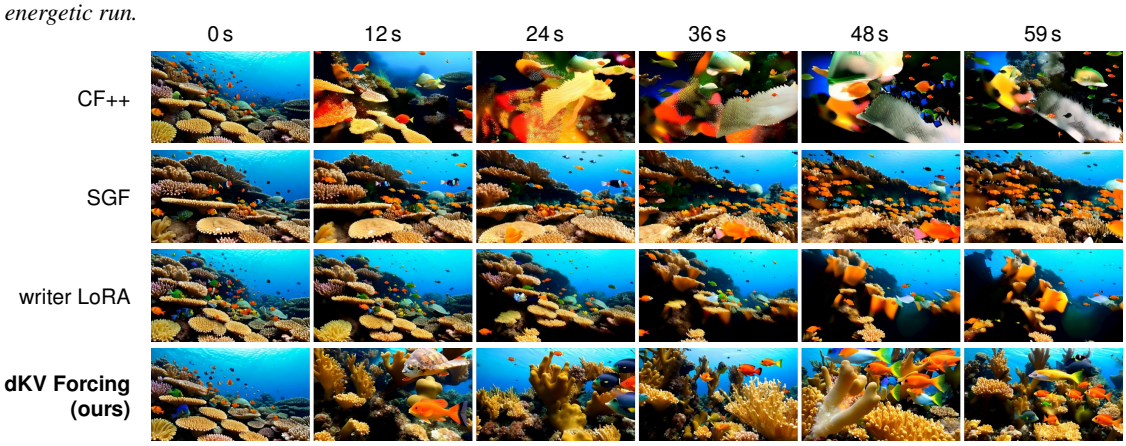


Figure 22. "Coral reef with fish and a turtle". Prompt: *A vibrant coral reef teeming with colorful fish and marine life, captured in a dynamic underwater scene. The reef is adorned with a variety of coral shapes and sizes, creating a lively and intricate landscape. Schools of fish dart between the corals, their vibrant hues ranging from electric blues and greens to fiery oranges and yellows. A schools of clownfish can be seen swimming among the anemones, while a majestic turtle glides gracefully nearby. The water has a clear, turquoise hue, allowing the viewer to see the rich biodiversity beneath the surface. The camera angle is slightly from below, emphasizing the vertical depth and the bustling activity of the reef.*

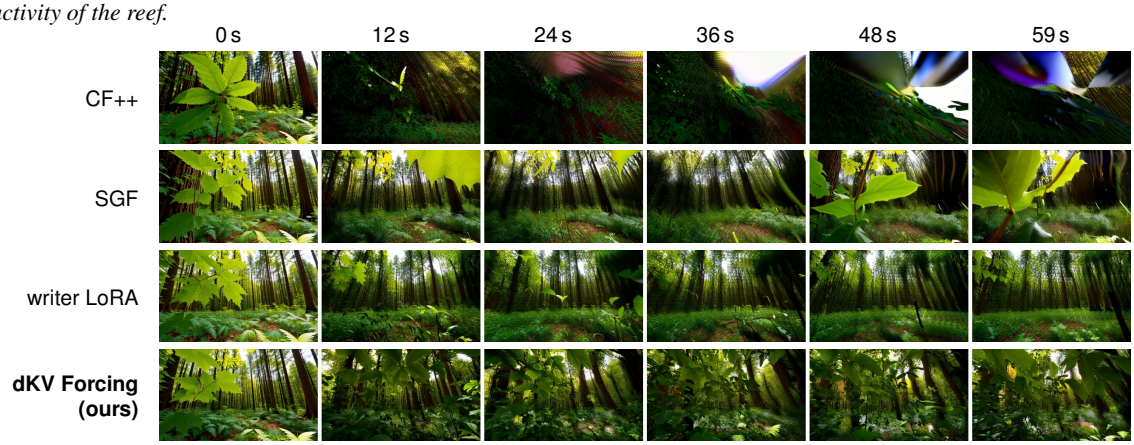


Figure 23. "Zoom-out from a leaf to the forest". Prompt: *A zoom-out from a single emerald green leaf on a tree to reveal the entire forest, capturing the vastness and diversity of the woodland. The scene starts with a close-up of the leaf, highlighting its intricate vein patterns and vibrant color. As the camera pulls back, it reveals a dense forest with towering trees, their trunks varying in size and shape, and a rich tapestry of foliage in various shades of green, brown, and gold. Ferns and wildflowers dot the forest floor, adding to the natural beauty. The sunlight filters through the canopy, casting dappled shadows on the ground. The forest exudes a serene and tranquil atmosphere, with birds chirping in the background. A wide-angle shot with a gradual pull-back.*



Figure 24. "Friends on the beach at sunset". Prompt: *A handheld shot capturing a group of friends laughing and playing on the beach at sunset. The friends, with joyful expressions, run and play among the waves, their laughter echoing across the golden sand. The camera follows them closely, emphasizing their lively movements and the joyous atmosphere. The background features a breathtaking sunset, with orange and pink hues blending into the horizon, and the gentle sea breeze blowing through. The sand is warm and the water sparkles under the setting sun. A dynamic and energetic scene, with the friends' silhouettes gradually becoming softer against the fading light.*



Figure 25. "Young man hugging a friend". Prompt: *A vibrant and lively photograph capturing a moment of genuine joy, depicting a young man with a radiant smile as he hugs a dear friend. His face is filled with happiness, with his eyes sparkling and cheeks flushed. He wears a casual shirt and jeans, and his hair is tousled, adding to the natural and relaxed feeling. His friend, equally joyful, returns the embrace with equal warmth. The background is a blurred setting, hinting at a park or a sunny outdoor space, with patches of green grass and trees in the distance. The photo has a warm and candid feel, with soft lighting and natural shadows. A close-up shot from a slightly angled perspective, emphasizing their heartfelt connection.*

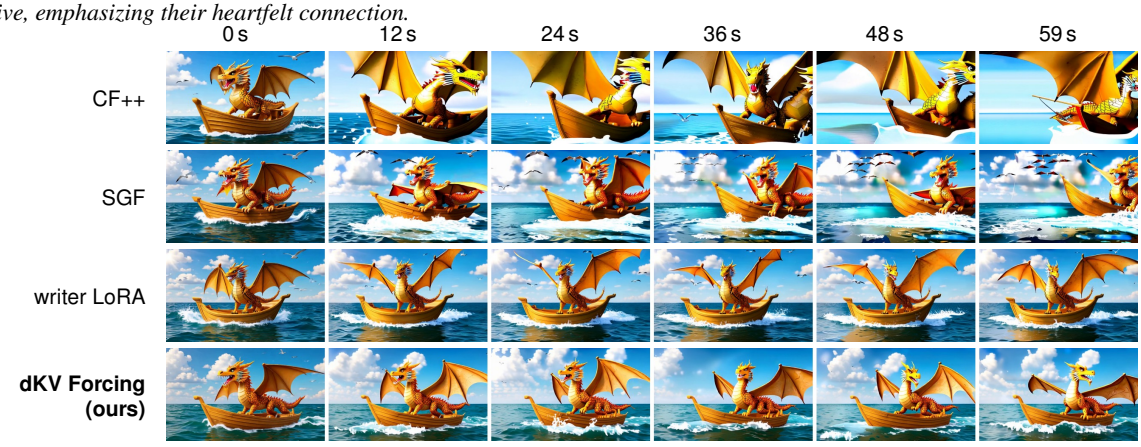


Figure 26. "Origami dragon riding a boat". Prompt: *A realistic style paper origami dragon riding a boat through waves, with intricate folds and textures. The dragon has a fierce expression, its eyes glowing with intensity, and its scales shimmering in the sunlight. It is perched on the edge of the boat, wings partially spread, ready to take flight. The boat bobs up and down with the waves, creating a dynamic motion. The water is choppy, with ripples and splashes around the boat, adding to the sense of movement. The background features a clear blue sky with fluffy clouds, and a few seagulls flying overhead. A mid-shot capturing the dragon's powerful stance and the boat's motion.*