BITWISE-IDENTICAL LATENT REUSE FOR CAUSAL VIDEO ENCODERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Causal video encoders recompute shared work across overlapping clips, but pixel overlap alone does not justify feature reuse: independently initialized clips differ in temporal support and sampling phase. We derive compatible reuse regions at multiple encoder depths. Shallow features become reusable earlier; deeper features become reusable later but avoid more computation. ClipStore combines both within a request, preserves request-local suffix state, and retains only planned reads using bounded rolling storage. On a frozen Wan2.1 encoder, all $4{,}944$ timed output comparisons across four videos match independent encoding exactly. On the reference plan, complete construction takes 9.9% less time and retains 21.9% fewer feature bytes than the faster fixed-cut baseline, with a $2.51\times$ speedup over independent encoding. Request-count, length, overlap, and grouping sweeps identify where source construction is amortized. The reported gain in a 50-step text-to-video pipeline is $1.006\times$; encoder reuse adds no output error when combined with DiT-side caching. Decoder continuation illustrates the complementary use of retained causal state to preserve an existing stream's outputs.

1 INTRODUCTION

A video latent cache stores encoded clips for later training or generation. Requests with adjacent starts overlap, yet each output must equal independent encoding of that clip. We ask which intermediate values can be shared across starts while preserving this contract.

Video generation models separate pixel encoding from generative modeling through learned latents (Rombach et al., 2022; Wan Team, 2025; Yang et al., 2025; Kong et al., 2024), and preprocessing systems already share transformation results across related tasks (Isenko et al., 2022; Haynes et al., 2021; Ye et al., 2025; Poms et al., 2018). Causal video encoders, however, obstruct a direct transfer of that idea, because every clip begins its own temporal state; consequently, cropping the source latent can change the requested result even when the underlying frames overlap exactly. A second obstacle pulls in the opposite direction: the encoder's temporal receptive field is wide—113 raw frames here, or 29 latent positions—so the region that a naive argument would call safe to copy is small enough that reuse looks not worth its complexity.

Reuse depends on the cumulative support at each cut, rather than only the whole encoder's field. A feature can be substituted when its dependency graph and sampling phase match the request. In the evaluated encoder, shallow features become compatible sooner but avoid less computation and occupy more bytes; deeper features become compatible later but avoid more work. A single request can exploit both.

ClipStore computes the request's boundary, reads shallow features while advancing its bridge state, then switches to deep features while preserving continuous suffix state. A rolling builder retains only planned reads with constant additional feature storage. Matching dependency graphs establishes mathematical equality; bitwise equality additionally depends on numerical execution and is checked for every timed output.

The contributions are:

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

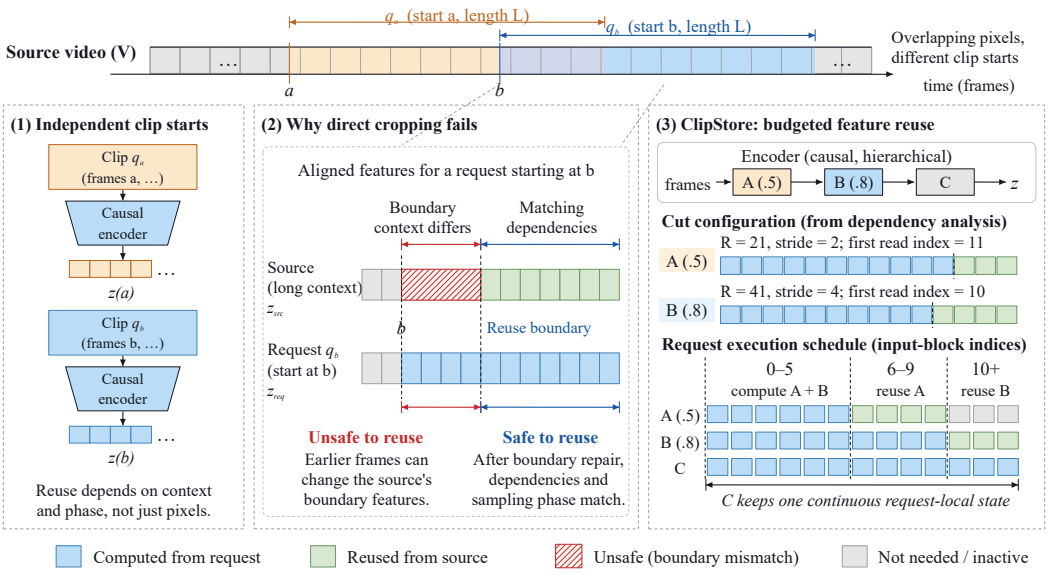

Figure 1: **Motivation and design of ClipStore.** (1) Overlapping requests q_a and q_b initialize independent causal states. (2) Cropping source features can fail near the request start b: their dependencies may include earlier frames. Reuse begins only when dependencies, initialization effects, and sampling phase match. (3) The non-origin request schedule computes A and B, then reads A while continuing B, and finally reads B. C maintains one continuous request-local state; only the middle stage consumes retained A positions. Receptive-field widths R are in frames. First-read indices refer to each cut's feature sequence; execution indices refer to input blocks (one initial frame, then four frames per block). Thus A feature index 11 corresponds to input block 6. Cell widths are schematic and do not represent measured time.

- A dependency-graph account of exact reuse for causal encoders that separates the two obstacles—initialization and phase—from the receptive field, and shows the field's decomposition across a cut to be what makes reuse usable at multiple depths (Section 3).
- A phase-aware two-depth execution rule preserving independent clip boundaries and continuous suffix state, with a plan-derived feature pack and rolling builder that retain only consumed positions under a constant storage bound, and a byte condition comparing its retained payload with a fixed cut (Section 4).
- An evaluation of exactness, complete construction cost, retained bytes, and allocation peaks against plan-pruned fixed-cut executors, across sweeps of request count, length, overlap, and grouping, plus decoder-side measurements (Sections 5–6).

2 Background and Motivation

The construction plan. A latent cache contains encoded outputs for requested clips. We call the set of source frames, starts, and lengths a construction plan. Grouping its requests by source exposes intermediate work shared during cache building, while later consumers simply load the final tensors. The separation already appears in existing pipelines: the inspected Musubi enumeration logic, executed on two video records under seven configurations, produced 426 construction requests and 852 subsequent fixed-cache reads (Musubi Tuner contributors, 2026). Final-output reuse already handles requests that share a start (a longer causal latent supplies shorter prefixes); different starts create separate initialization boundaries. ClipStore shares intermediate values across those boundaries while constructing the distinct final outputs.

Overlap and independent boundaries. Consider two 129-frame clips beginning at source frames 40 and 60. They share 109 frames, yet the second starts its temporal history at frame 60, while the first has already processed twenty frames by then. A causal encoder can therefore assign different early features to the shared pixels: overlap alone does not justify copying the latent region. At a

prefix with finite temporal support, by contrast, earlier history eventually leaves the dependency graph, and interior features then depend on the same pixels and operators as independent encoding; temporal subsampling further requires that the feature phase match.

Figure 2 measures this boundary effect at three depths. Across eight 129-frame cases at each level, directly cropped source values have nonzero error near the request start, and become exactly zero from frame offset 20 at cut A, 40 at cut B, and 112 at the posterior mean itself—the naive strategy of encoding a longer history and cropping the latent. These are the availability times of the three stages Section 3 derives, and the gaps between them are the opportunity: A already supplies compatible values while B still needs the request's own computation, and both are compatible long before the composed field clears at 112.

Choosing what to retain. A fixed shallow cache becomes usable early but leaves downstream work; a fixed deep cache saves more but arrives later. Section 4 shows a request needs the shallow depth only between the two boundaries, which is what makes selective retention pay. Length-dependent sampling, spatial crops, and changed weights define separate sharing groups, which preprocessing systems can supply upstream (DiffSynth-Studio contributors, 2026; Hugging Face, 2026; Murray et al., 2021; Leclerc et al., 2023; Isenko et al., 2022).

3 Exact Reuse under Depth Stratification

We fix a frozen encoder and input semantics throughout; the locality arguments below are a correctness contract, not a new claim about convolutions. The claim is structural: exact reuse is a property of the encoder's *depth structure*, and the reuse depth decides what it costs and what it displaces.

3.1 Objects and two obstacles

Let X be a decoded, spatially transformed frame sequence and $q = (a, L, \sigma, \theta)$ a request that independently encodes $X[a : a + L)$, with σ its sampling semantics and θ the frozen encoder, precision, and boundary convention; we use the deterministic posterior mean. A source object begins at origin $o+p$ for phase p; at cut ℓ its feature sequence is $\bar{H}_{\ell,p}$ and the request produces $H_\ell(q)$. Let $d = a-o$. For a causal prefix with temporal stride j_ℓ and width R_ℓ, an output anchored at time t has support

$$D_\ell(t) \subseteq [t - (R_\ell - 1), t]. \tag{1}$$

This is a structural upper bound, so initialization branches and first-frame bypasses must also have left the graph before it applies; spatial attention within a frame does not enlarge temporal support.

Under these fixed semantics, two sufficient interior conditions separate a compatible source feature from a potentially incompatible one. The first is *support*: the feature's dependency subgraph must lie inside the request, so the request's own left boundary has left the graph. The second is *phase*: the feature must occupy the same temporal sampling slot the request would use. Depth, channel width, parameter count, and the global field's magnitude enter only through these two conditions—which is what makes reuse a family of valid boundaries rather than a single decision.

Proposition 1 (Compatible interior reuse)**.** *Suppose a source and request feature have identical operator dependency graphs, input values, parameters, and temporal sampling phase. If all leaves of the relevant temporal graph lie inside the request and no differing initialization branch remains, their mathematical values are equal.*

Proof. Induction through the directed acyclic dependency graph from the agreeing input leaves; causality excludes dependence on the right boundary. □

For regular positions, the mapping $p = d \bmod j_\ell$, $n = (d - p)/j_\ell + k$ locates request feature k in source phase p, and containment $D_\ell(a + j_\ell k) \subseteq [a, a + L)$ gives a conservative interior condition, although exact origin matches may be reusable earlier.

3.2 Depth controls both conditions

Apply the propagation rule for a causal convolution with kernel k, dilation r, and stride s,

$$R' = R + (k - 1)rj, \qquad j' = sj, \tag{2}$$

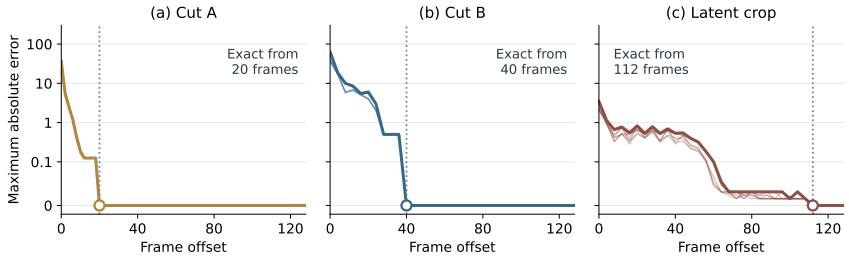

Figure 2: Measured boundary mismatch between cropped source values and independent clip values at three depths: cut A, cut B, and the posterior mean (latent crop). Each panel has eight cases: two videos, four starts, length 129. Thin lines show individual cases; the thick line is their pointwise maximum absolute error. The axis is linear near zero and logarithmic above 0.1. The three depths become exactly equal at frame offsets 20, 40, and 112.

taking the union across residual branches. Support width and sampling stride are nondecreasing with depth for the encoder considered here. Because R_ℓ shrinks toward the input, an earlier cut satisfies support after a shorter repair, but leaves more downstream computation and has larger features in this encoder. j_ℓ likewise shrinks, so an earlier cut has a finer phase lattice and matches more requests, at the cost of larger features.

Because neither depth necessarily dominates, a request can use a monotone sequence of cuts, each contributing the region where it is already compatible, with the suffix holding continuous request-local state across transitions. Such an executor reads at depth ℓ_i where ℓ_i is compatible and the next deeper cut is not yet. When compatibility regions are nested suffix intervals, as in our two-cut schedule, the transition regions are intervals; the executor stores their union across the plan. Section 4 instantiates this with two depths, though the argument is not specific to two.

3.3 State continuity at a reuse transition

The encoder suffix processes feature blocks using

$$(S_{i+1}, Y_i) = F(S_i, H_i), \qquad S_0 = S_{\text{init}}(q). \tag{3}$$

Replacing a block with an equal source block preserves the next suffix state and output, so induction preserves the complete request, provided the suffix is initialized once and advances in the original block order. Importing source state or resetting it at a transition would violate this —exactly the two failure modes a naive splice introduces. Mathematical and bitwise equality are nevertheless separate contracts; our implementation preserves the original temporal blocks, keeps native BF16 features, and validates every output against the pinned independent encoder (Appendix B).

3.4 Instantiating the boundaries in Wan

Starting from $(R, j) = (1, 1)$ and applying Equation 2 to the inspected encoder, the input convolution gives $(3, 1)$; `.5` yields $(R, j) = (21, 2)$ and `.8` yields $(41, 4)$; the remaining blocks give $(113, 4)$ at the posterior mean (full chain in Appendix A). The stratification is explicit: a 113-frame global field decomposes into stage-local supports of 21 raw frames for A, 11 A positions for B, and 19 B positions for C. With stage strides 2, 2, and 1, these compose as $21 + 2(11 - 1) + 4(19 - 1) = 113$. The cumulative supports at the three depths are therefore 21, 41, and 113 raw frames. Figure 2 measures all three availability times: cropped values become exact at frame offsets 20 at A, 40 at B, and 112 at the posterior mean, on both videos and all four phases. The global field is thus the wrong denominator: each stage is bounded by its own support, not the composed one.

Under the original block schedule these bounds place the first wholly reusable block at index eleven for A and ten for B (Table 3, Appendix A). Crucially, B's 37-frame logical repair does not recompute A throughout: A runs only through its own 21 frames, and the rest of B's repair consumes compatible A source features—the computation the middle stage saves, and the first concrete instance of the stratification argument.

4 BUDGETED TWO-DEPTH REUSE

Because the stratification above admits several valid boundaries, the practical question is which ones to retain for a given plan. We therefore partition the encoder into A, ending at `.5`; B, the bridge through `.8`; and C, the remainder, so that source features have two A phases and four B phases; Figure 3 shows how source construction, selective storage, and request execution connect. The full executor is Algorithm 1 (Appendix B).

Request execution. A non-origin request follows three stages (Figure 1(3)). Blocks 0–5 compute A and B from the request's own boundary, one initial frame followed by four-frame blocks; blocks 6–9 then read A features while continuing the request's B state; and from block 10 the request reads B features and executes C. Each prefix state is released at its transition, while C processes every block with continuous request-local state, so its outputs concatenate before the unchanged posterior projection. Matching origins permit earlier reuse; short requests stop at their requested length.

Storage derived from the request plan. For relative start d_q and length $L_q = 4m_q+1$, the phases are $p_q = d_q \bmod 2$ and $r_q = d_q \bmod 4$. A non-origin request reads the inclusive shallow interval

$$I_A(q) = [\lfloor d_q/2 \rfloor + 11, \lfloor d_q/2 \rfloor + 2\min(m_q, 9)], \tag{4}$$

empty when $m_q < 6$ or the request matches a B origin ($d_q < 4$). The deeper interval is

$$I_B(q) = \begin{cases} [0, m_q], & d_q < 4, \\ [\lfloor d_q/4 \rfloor + 10, \lfloor d_q/4 \rfloor + m_q], & d_q \geq 4, \end{cases} \tag{5}$$

with the non-origin case empty when $m_q < 10$. Merging per-phase intervals gives the read unions $K_p^A = \bigcup_{q:p_q=p} I_A(q)$ and $K_r^B = \bigcup_{q:r_q=r} I_B(q)$. Each nonempty set receives a preallocated native tensor; original indices map to sorted packed slots, and reads then reconstruct contiguous blocks with their original values, temporal order, and operator shapes.

Proposition 2 (Read-set sufficiency)**.** *For a fixed two-depth schedule with source features built before service, $\{K_p^A, K_r^B\}$ serves every planned feature read. If native positions are stored without recomputation or cross-position compression, removing any selected position makes at least one read incomplete.*

Proof. Every read contributes its indices to a union, so its block is available; conversely, every selected index belongs to a read that requires its value. □

Storage and amortization laws. With u_A, u_B bytes per position ($u_B = u_A/2$ here), retained payload is $M_{\text{keep}} = u_A \sum_p |K_p^A| + u_B \sum_r |K_r^B|$. With F fixed A's retained count, $K_A = \sum_p |K_p^A|$, $K_B = \sum_r |K_r^B|$, two-depth storage fits within fixed A's payload exactly when

$$K_A + K_B/2 \leq F; \tag{6}$$

the reference plan gives 150 versus 192 A-equivalent positions, and a budget M is testable before encoding via $u_A K_A + u_B K_B \leq M$. For compute, let $E(q)$ be the cost of encoding q independently, $P_\ell(q)$ its prefix repair, $U_\ell(q)$ suffix execution, $I_\ell(q)$ read and transfer, and B_ℓ source construction and any additional intermediate writes. With all costs in time and common final-output writes excluded from both sides, reuse beats independent encoding when

$$\sum_{q \in Q} \big[E(q) - P_\ell(q) - I_\ell(q) - U_\ell(q)\big] > B_\ell. \tag{7}$$

Shallow reads can reduce boundary recomputation while deep reads avoid bridge computation later in the request. Their net benefit depends on source construction, read costs, and the request plan; it need not exceed that of either fixed cut. The byte count follows from the plan alone, whereas the runtime inequality needs cost estimates. When the request set is not known in advance, the same accounting applies per cross-group reuse opportunity: over K separated groups, retaining a source object is worthwhile only if

$$\sum_{k=2}^{K} A_k > W + \sum_{k=1}^{K} I_k + D, \tag{8}$$

where A_k is the prefix construction avoided in group k, W the initial durable-write cost, I_k later loading costs, and D the opportunity cost of retention.

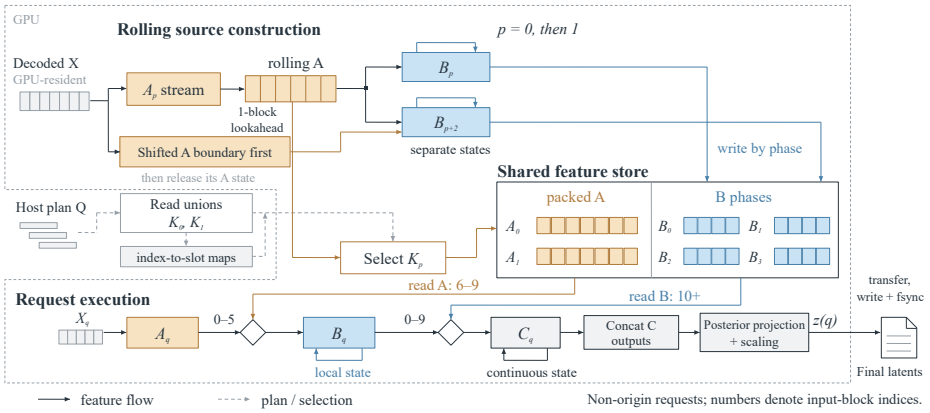

Figure 3: **Construction and request execution in ClipStore.** Host-side plan analysis determines the shallow read unions (K_0, K_1 in the diagram) and index-to-slot maps. For each parity p, the builder first completes the shifted phase's A boundary, then streams canonical A into separate B_p and B_{p+2} states. It packs selected A positions and releases consumed rolling tensors. The implementation also packs the required B positions directly into their preallocated destinations; analogous B index maps are omitted for clarity. The shared feature store resides on the GPU. Diamonds select computed or reused features according to the non-origin request's input-block index; they do not merge states. C preserves one request-local state, and its outputs are concatenated before posterior projection and scaling. Final latents are transferred and written with `fsync`. Matching source origins may reuse earlier than the schedule shown. Inputs are already decoded and GPU-resident, as in the timed experiment.

Rolling construction. The builder processes parity pairs $(A_0; B_0, B_2)$ and $(A_1; B_1, B_3)$ sequentially. For each pair, the shifted B phase first computes its own A boundary through block five and releases that state, then the canonical A stream supplies both B phases with one-block lookahead. Selected positions are copied into preallocated packs and released after consumption, so at most two canonical A blocks wait in the window. Deduplicating aliases gives

$$M_{\text{features,peak}} \le M_{\text{keep}} + 6u_A + u_B, \tag{9}$$

an additive 62.30 MB at the evaluated shape, independent of source length (Appendix B); fixed B uses the same builder with an empty A pack.

One executor, three policies. Fixed A enables only shallow reads, fixed B only deep reads, and ClipStore both; all three share source construction, packing, phase pruning, and the final-output path, so comparisons isolate representation depth. On the reference plan this removes 216 ordinary B calls against fixed A and 48 A calls against fixed B (Appendix B).

5 Evaluation

We evaluate exactness, complete construction cost, and the relation between request structure and retained feature space; the primary comparison isolates representation depth, and the sweeps that follow locate where selective depth pays for itself.

Setup. Experiments use NVIDIA A100-PCIE-40GB GPUs, PyTorch 2.5.1, and the frozen Wan2.1 encoder at batch size one, on four public videos decoded to 514 frames at 480×832 (Appendix C). The reference plan has sixteen 129-frame requests across four starts and four phases. We compare four executors: **Independent** calls the official encoder per request; **Fixed A** retains only planned A positions; **Fixed B** retains only planned B positions and shares ClipStore's rolling builder; and **ClipStore** retains both read unions and changes depth within each request. All reuse executors receive the full plan, prune unused phases and positions, and share the packed layout and final-output path; the timer spans source-range copying through all final writes and `fsync`. The reference plan has three repetitions per video; each count, length, and spacing sweep point has one observation per video. Temporal grouping has two repetitions on `vtest`; compressed-input construction has one run per video. We report means with full observed ranges.

Exactness. Every timed output is compared with an independently produced official-encoder reference under zero-tolerance elementwise equality: 252 timed groups, 4,944 comparisons, all exact (3,984 of them from the 216-group construction matrix alone). Post-timing file comparisons verify all 3,072 reference-plan data shards byte for byte; deliberately shortening repair or choosing the wrong phase produces unequal outputs (Figure 2).

5.1 Complete construction cost

Table 1 reports the reference plan (bar chart in Appendix D, Figure 5). ClipStore takes 28.497 seconds versus 31.637 (Fixed A), 34.059 (Fixed B), and 71.626 (Independent): 9.93% less time than Fixed A and 16.33% less than Fixed B including shared source construction, and $2.51\times$ over independent encoding. All twelve paired observations favor ClipStore (Fixed-A/ClipStore ratio $1.110\times$, range $[1.107, 1.113]$).

The two fixed cuts reveal separate benefits: against Fixed A, ClipStore saves 7.046 seconds of request computation for 3.919 extra seconds of B build; against Fixed B, build cost is nearly identical while request computation falls from 18.018 to 12.468 seconds, isolating shallow reuse during the deep-boundary repair (Appendix D). The shallow pack occupies 0.518 GB and the deep pack 0.920 GB—together 1.438 GB, 21.9% less than Fixed A's 1.840 GB. The tracked feature-construction peak is 1.500 GB, matching the rolling allowance of Equation 9, and the whole-group CUDA peak falls 3.21% against Fixed A.

5.2 Scaling, overlap, and grouping

Figure 4 varies request count and clip length with all starts away from source origins. At sixteen requests ClipStore takes 32.05 seconds versus 33.31 (Fixed A); at thirty-two, 54.35 versus 59.64 ($1.097\times$ paired). At four and eight requests Fixed A is faster, because a small plan does not consume enough deep features to repay their construction. This is the runtime tradeoff in Equation 7; Equation 6 predicts retained bytes, not elapsed time. Lengthening clips leaves the middle-stage A interval fixed after eight positions: at 257 frames, ClipStore takes 45.29 seconds versus 52.12 (Fixed A) and 52.66 (Fixed B) with 28.9% less storage—more of the output lies beyond the deep boundary; a 33-frame clip never reads B, and at 65–81 frames the extra B build has not yet paid off.

A spacing sweep at fixed count and length (Appendix D) shows the complementary effect: at spacing 4, ClipStore takes 28.64 seconds against Fixed A's 30.61 and retains 0.786 versus 1.169 GB, while at spacing 40 the times converge to 36.43 and 36.81. Denser overlap therefore shares repair inputs, whereas dispersed starts require a longer continuously computed source prefix. Table 2 extends the comparison to three further structures: dense reuse across 64 requests yields $1.212\times$ over the faster fixed cut, a single-phase plan $1.175\times$, and a mixed-length plan cuts Fixed A's payload by 32.7% at 1.4% more time.

Finally, partitioning a thirty-two-request plan into groups of 4–32, with each group building and releasing its own source, ClipStore takes 51.01 seconds in a single group against 58.28 and 63.94 for the fixed cuts; even under a tight 1.5 GB payload budget it reaches 57.42 seconds, where the fastest feasible fixed configuration takes 63.94 (Appendix D); grouping and reuse depth must be chosen together.

From compressed video to final latents. Charging decoding, resizing, RGB normalization, and initial GPU transfer as well, ClipStore completes the reference plan in 31.00 seconds versus 34.21 (Fixed A) and 36.69 (Fixed B), preserving a 9.4% reduction (Appendix D); all 192 outputs again match independent encoding.

6 Discussion

Where the stage-level gain matters end to end. The $2.51\times$ factor belongs to encode-dominated construction: in 50-step Wan2.1-T2V-1.3B generation the encoder is under 2% of latency, so ClipStore yields $1.006\times$ end to end, with a combined reported speedup of $2.015\times$ when DiT-side caching is enabled (DiT-side caching alone: $1.993\times$), whose $1.99\times$ costs 15–21 dB deviation while ours is bitwise-lossless. Under few-step distilled DiTs the encoder share rises to roughly 13%, and in

Table 1: Reference-plan construction and memory (means of 12 observations). Peak GPU is the maximum allocated through the whole group; features count native A/B payload. All methods produce identical final files.

Executor	Complete (s)	Build (s)	Requests (s)	Writes (s)	Features (GB)	Peak GPU (GB)
Independent	71.626	0.000	70.613	1.010	0.000	10.678
Fixed A	31.637	11.130	19.513	0.990	1.840	12.508
Fixed B	34.059	15.045	18.018	0.993	0.920	11.591
ClipStore	**28.497**	15.049	**12.468**	0.977	1.438	12.107

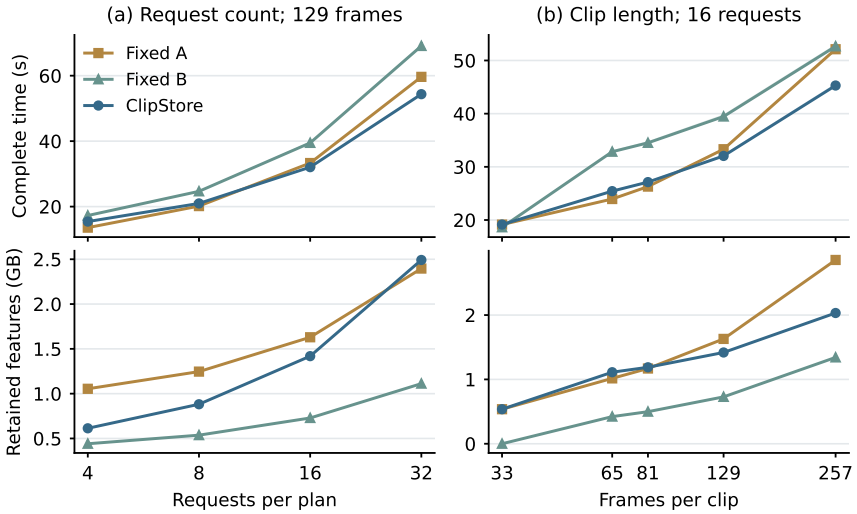

Figure 4: Scaling with (a) request count and (b) clip length. Upper panels show complete time; lower panels show native retained payload. Lines are four-video means, bands are observed ranges. All starts are non-origin and every method receives the same plan and source extent.

latent-cache construction or multi-shot preprocessing—the workloads of Section 5—encoding is the pipeline.

The decoder inverts the economics. The stratification argument is mechanical—Equation 2 computes every cut's availability time and phase lattice from the module graph alone—so it applies unchanged to the decoder, with one simplification and one reversal: the decode driver feeds one latent per chunk, so every crop is chunk-aligned, the phase condition is trivial, and only support remains. Equation 2 through the decoder gives a 37.5-position field—horizon 38—confirmed stage by stage (Appendix E). But the cumulative supports at the temporal-upsample boundaries are 22, 30, and 38: reach accrues in the cheap latent-rate stages while compute concentrates in the high-resolution suffix, whose own contribution is 6.5 positions, so interior feature reads displace little decoder computation. What transfers instead is state continuity (Equation 3): snapshotting the causal caches at a chunk boundary and resuming reproduces the remaining source-decode frames bitwise at zero repair, 1.19–2.85× faster than re-decoding from the safe horizon (Appendix E). The encoder rewards interior reads; the decoder rewards retained state.

Scope. Our evaluation covers one autoencoder family at one precision on a single device, and we do not build a decoder-side planner. The bitwise contract holds for the recorded execution configuration; other kernels, batch shapes, or precision modes require revalidation. The plan is known in advance, so the online case is bounded by Equation 8 rather than demonstrated; requests resemble cache-construction and multi-shot traffic, not a replayed trace; and on dispersed plans peak memory exceeds fixed-A reuse (Section 5).

Table 2: Additional plans: mean complete seconds / retained GB over four videos. Dense uses 64 requests; single-phase and mixed use sixteen.

Plan	Fixed A	Fixed B	ClipStore
Dense	98.83 / 1.629	111.13 / 0.728	81.54 / 1.476
Single phase	29.15 / 1.093	32.18 / 0.254	24.81 / 0.906
Mixed	39.15 / 2.741	46.66 / 1.174	39.68 / 1.845

7 Related Work

Video representations and materialization. Latent diffusion separates learned compression from generative modeling (Rombach et al., 2022), and Wan, CogVideoX, and HunyuanVideo extend it to video with causal spatiotemporal encoders (Wan Team, 2025; Yang et al., 2025; Kong et al., 2024). Dependency analysis and materialization are established for such pipelines: Scanner selects required computation from bounded-state operators (Poms et al., 2018), VSS manages overlapping materializations (Haynes et al., 2021), SAND merges identical preprocessing objects under a budget (Ye et al., 2025), and PRESTO profiles the offline/online tradeoff (Isenko et al., 2022). Training pipelines such as tf.data, FFCV, NoPFS, and SOLAR likewise exploit known access schedules (Murray et al., 2021; Leclerc et al., 2023; Dryden et al., 2021; Sun et al., 2022). ClipStore instead exposes the encoder's *internal* sharing relation—phase and initialization determine which positions remain compatible—and uses its plan to share the neural computation itself.

Reusing local neural computation. DeepCache propagates reusable regions through layers by kernel, stride, and padding (Xu et al., 2018); CBinfer updates changed convolution positions (Cavigelli et al., 2017); Co3D reuses temporal state in continual video inference and studies padding and initialization (Hedegaard & Iosifidis, 2022). Region propagation and boundary-aware execution are thus well established, but all three assume a shared input history that independent clips lack; ClipStore carries the model to that harder setting, each request repairing its own boundary and then reading progressively deeper compatible features under its own suffix state. MCUNetV2 and StreamNet retain overlapping intermediates under memory constraints (Lin et al., 2021; Zheng et al., 2023), and CacheBlend combines cached non-prefix context with empirically chosen KV recomputation (Yao et al., 2025); ClipStore, by contrast, derives its reuse boundary from finite temporal dependencies and matching initialization graphs.

Materialization under memory constraints. The red-blue pebble model (Hong & Kung, 1981), rematerialization systems (Chen et al., 2016; Jain et al., 2020; Kirisame et al., 2021; Bartan et al., 2023; Zhao et al., 2023), and residency managers (Rhu et al., 2016; Huang et al., 2020) show why retained outputs, live temporaries, and execution cost must be considered together; ClipStore applies the same accounting to phase-indexed features.

8 Conclusion

ClipStore reuses compatible shallow and deep features within independently initialized clips while maintaining request-local suffix state. Cumulative support and temporal phase determine when each depth becomes reusable; read unions determine what to retain. On the reference plan, construction takes 9.93% less time and retains 21.9% fewer feature bytes than the faster fixed cut. All 4,944 timed comparisons match independent encoding exactly. The sweeps show when these savings repay source construction; decoder continuation illustrates the complementary role of retained causal state.

Ethics statement

This work studies the systems properties of a frozen, publicly released video autoencoder on four public video samples; it involves no human subjects, no personal data, and no new generative capability. Exact reuse reduces computation for the tested workloads; energy consumption was not measured.

AI USE STATEMENT

Large language model assistants were used during manuscript revision for language editing, literature and citation checking, checking mathematical exposition, and interpreting existing experimental summaries. These revisions included corrections to receptive-field accounting and cost comparisons. The authors are responsible for the final text, claims, and artifacts.

REPRODUCIBILITY STATEMENT

Every timed output is gated bitwise against the official encoder inside the harness, and runs record source revisions, script hashes, request plans, and per-observation timings (Section 5; Appendix B and Appendix C give the execution and platform details, including the autocast boundary required to reproduce bitwise equality). The artifact includes the executor, planner, measurement harness, analysis, and figure scripts, together with input provenance (URLs, hashes, and preprocessing) for the four public videos. Anonymous repository: `https://anonymous.4open.science/status/ClipStore-6D88`.

REFERENCES

Burak Bartan, Haoming Li, Harris Teague, Christopher Lott, and Bistra Dilkina. Moccasin: Efficient tensor rematerialization for neural networks. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 1826–1837. PMLR, 2023. URL https://proceedings.mlr.press/v202/bartan23a.html.

Lukas Cavigelli, Philippe Degen, and Luca Benini. CBinfer: Change-based inference for convolutional neural networks on video data. In *Proceedings of the 11th International Conference on Distributed Smart Cameras*, pp. 1–8, 2017. doi: 10.1145/3131885.3131906. URL https://arxiv.org/abs/1704.04313.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016. doi: 10.48550/arXiv.1604.06174. URL https://arxiv.org/abs/1604.06174.

DiffSynth-Studio contributors. DiffSynth-Studio: Data operator source code. https://github.com/modelscope/DiffSynth-Studio/blob/c458cb42ab1ee838bff85c6546e14bb01c3571e9/diffsynth/core/data/operators.py, 2026. Pinned source revision dated September 14, 2026; lines 137–195.

Nikoli Dryden, Roman Böhringer, Tal Ben-Nun, and Torsten Hoefler. Clairvoyant prefetching for distributed machine learning I/O. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2021. doi: 10.1145/3458817.3476181. URL https://arxiv.org/abs/2101.08734.

Brandon Haynes, Maureen Daum, Dong He, Amrita Mazumdar, Magdalena Balazinska, Alvin Cheung, and Luis Ceze. VSS: A storage system for video analytics. In *Proceedings of the 2021 International Conference on Management of Data*, pp. 685–696, 2021. doi: 10.1145/3448016.3459242.

Lukas Hedegaard and Alexandros Iosifidis. Continual 3D convolutional neural networks for real-time processing of videos. In *Computer Vision – ECCV 2022*, pp. 369–385, 2022. doi: 10.1007/978-3-031-19772-7_22. URL https://www.ecva.net/papers/eccv_2022/papers_ECCV/papers/136640364.pdf.

Jia-Wei Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, pp. 326–333, 1981. doi: 10.1145/800076.802486.

Chien-Chin Huang, Gu Jin, and Jinyang Li. SwapAdvisor: Pushing deep learning beyond the GPU memory limit via smart swapping. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1341–1355,

2020. doi: 10.1145/3373376.3378530. URL https://par.nsf.gov/servlets/purl/10191573.

Hugging Face. finetrainers: Video processing source code. https://github.com/huggingface/finetrainers/blob/7e9257aae285c2fd665b39faf5c4109762762438/finetrainers/functional/video.py, 2026. Pinned source revision dated April 8, 2026; lines 32–92.

Alexander Isenko, Ruben Mayer, Jeffrey Jedele, and Hans-Arno Jacobsen. Where is my training bottleneck? hidden trade-offs in deep learning preprocessing pipelines. In *Proceedings of the 2022 International Conference on Management of Data*, pp. 1825–1839, 2022. doi: 10.1145/3514221.3517848.

Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gholami, Pieter Abbeel, Kurt Keutzer, Ion Stoica, and Joseph E. Gonzalez. Checkmate: Breaking the memory wall with optimal tensor rematerialization. In I. Dhillon, D. Papailiopoulos, and V. Sze (eds.), *Proceedings of Machine Learning and Systems*, volume 2, pp. 497–511, 2020. URL https://proceedings.mlsys.org/paper_files/paper/2020/file/0b816ae8f06f8dd3543dc3d9ef196cab-Paper.pdf.

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. URL https://openreview.net/forum?id=Vfs_2RnOD0H.

Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, et al. HunyuanVideo: A systematic framework for large video generative models. *arXiv preprint arXiv:2412.03603*, 2024. doi: 10.48550/arXiv.2412.03603. URL https://arxiv.org/abs/2412.03603.

Guillaume Leclerc, Andrew Ilyas, Logan Engstrom, Sung Min Park, Hadi Salman, and Aleksander Mądry. FFCV: Accelerating training by removing data bottlenecks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12011–12020, 2023. URL https://openaccess.thecvf.com/content/CVPR2023/html/Leclerc_FFCV_Accelerating_Training_by_Removing_Data_Bottlenecks_CVPR_2023_paper.html.

Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song Han. Memory-efficient patch-based inference for tiny deep learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 2346–2358, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/1371bccec2447b5aa6d96d2a540fb401-Abstract.html.

Derek G. Murray, Jiří Šimša, Ana Klimovic, and Ihor Indyk. tf.data: A machine learning data processing framework. *Proceedings of the VLDB Endowment*, 14(12):2945–2958, 2021. doi: 10.14778/3476311.3476374. URL https://www.vldb.org/pvldb/vol14/p2945-klimovic.pdf.

Musubi Tuner contributors. Musubi Tuner: Dataset enumeration and bucket source code. https://github.com/kohya-ss/musubi-tuner/tree/e0cbd8f3dfe38365b10f8bc790b980f8894e8ba1/src/musubi_tuner/dataset, 2026. Pinned source revision dated August 13, 2026; files image_video_dataset.py and bucket.py.

Alex Poms, Will Crichton, Pat Hanrahan, and Kayvon Fatahalian. Scanner: Efficient video analysis at scale. *ACM Transactions on Graphics*, 37(4):138:1–138:13, 2018. doi: 10.1145/3197517.3201394. URL https://graphics.stanford.edu/papers/scanner/.

Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W. Keckler. vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 1–13, 2016. doi: 10.1109/MICRO.2016.7783721. URL https://arxiv.org/abs/1602.08124.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022. doi: 10.1109/CVPR52688.2022.01042. URL https://openaccess.thecvf.com/content/CVPR2022/html/Rombach_High-Resolution_Image_Synthesis_With_Latent_Diffusion_Models_CVPR_2022_paper.html.

Baixi Sun, Xiaodong Yu, Chengming Zhang, Jiannan Tian, Sian Jin, Kamil Iskra, Tao Zhou, Tekin Bicer, Pete Beckman, and Dingwen Tao. SOLAR: A highly optimized data loading framework for distributed training of CNN-based scientific surrogates. *arXiv preprint arXiv:2211.00224*, 2022. doi: 10.48550/arXiv.2211.00224. URL https://arxiv.org/abs/2211.00224.

Wan Team. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025. doi: 10.48550/arXiv.2503.20314. URL https://arxiv.org/abs/2503.20314.

Wan-Video. Wan2.1 VAE implementation. https://github.com/Wan-Video/Wan2.1/blob/9737cba9c1c3c4d04b33fcad41c111989865d315/wan/modules/vae.py, 2026. Pinned source revision dated March 5, 2026.

Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. DeepCache: Principled cache for mobile deep vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pp. 129–144, 2018. doi: 10.1145/3241539.3241563.

Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, Da Yin, Yuxuan Zhang, Weihan Wang, Yean Cheng, Bin Xu, Xiaotao Gu, Yuxiao Dong, and Jie Tang. CogVideoX: Text-to-video diffusion models with an expert transformer. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2408.06072.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 94–109, 2025. doi: 10.1145/3689031.3696098.

Juncheol Ye, Seungkook Lee, Hwijoon Lim, Jihyuk Lee, Uitaek Hong, Youngjin Kwon, and Dongsu Han. SAND: A new programming abstraction for video-based deep learning. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, pp. 589–605, 2025. doi: 10.1145/3731569.3764847.

Xunyi Zhao, Théotime Le Hellard, Lionel Eyraud-Dubois, Julia Gusak, and Olivier Beaumont. Rockmate: an efficient, fast, automatic and generic tool for re-materialization in PyTorch. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 42018–42045. PMLR, 2023. URL https://proceedings.mlr.press/v202/zhao23b.html.

Hong-Sheng Zheng, Yu-Yuan Liu, Chen-Fong Hsu, and Tsung Tai Yeh. StreamNet: Memory-efficient streaming tiny deep learning inference on the microcontroller. In *Advances in Neural Information Processing Systems*, volume 36, pp. 37160–37172, 2023. doi: 10.52202/075280-1615. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/7526508f11bbe0a123af62b9dab1fbe1-Abstract-Conference.html.

A Receptive-field derivation for the Wan encoder

Starting from $(R, j) = (1, 1)$ and applying Equation 2 layer by layer to the inspected encoder—two kernel-three convolutions per residual block, temporal downsampling configuration $[\text{false}, \text{true}, \text{true}]$—the input convolution gives $(3, 1)$. The first downsampling group's residual blocks and spatial resample leave (R, j) at $(11, 1)$; the first temporal downsample doubles the stride. Cut `.5` (module `downsamples[5]`) yields $(R, j) = (21, 2)$; the bridge through `.8` (`downsamples[8]`) yields $(41, 4)$; the two subsequent residual blocks, two middle residual blocks, and the final kernel-three convolution give $(113, 4)$ at the posterior mean. The middle attention is spatial, and the posterior projection and scaling are temporally pointwise. Composing the per-stage supports recovers the global field: A spans 21 raw frames at stride 2, B spans 11 A positions at stride 2, and C spans 19 B positions at stride 1. Thus $R_{AB} = 21 + 2(11 - 1) = 41$ and $R_{ABC} = 41 + 4(19 - 1) = 113$ raw frames. The field was additionally confirmed by perturbation: injecting a large change at one raw frame alters latents up to 28 positions later, matching 113 frames at stride 4.

Under the original block schedule (one initial frame, then four-frame blocks), the first structurally eligible feature at both cuts has local index ten. At A, index ten shares an execution block with an earlier unsafe feature, so the first wholly reusable A block begins at index eleven; at B, the first wholly reusable output block begins at index ten (Table 3). Shortening the repair below these boundaries or selecting the wrong phase produces measurably unequal outputs, with the shortest failing configuration differing by a single BF16 unit in the last place—the boundaries are tight, not conservative.

B Executor, implementation, and accounting

Algorithm. Algorithm 1 uses raw blocks $I_0 = [0, 1)$ and $I_i = [4i - 3, 4i + 1)$ for $i > 0$, with A indices $U_0 = \{0\}$ and $U_i = \{2i - 1, 2i\}$. The B predicate requires a B-using policy and either $d < 4$ or $i \geq 10$; otherwise the A predicate requires an A-using policy and either $d < 2$ or $i \geq 6$. Fixed A enables only A, fixed B only B, and ClipStore both; their read sets enter one packed builder. Origin equality uses relative start rather than congruence alone: a later request with the same phase still repairs its boundary.

Encoder and interface. We use Wan2.1 VAE revision `9737cba9c1c3` (Wan-Video, 2026), FP32 weights, BF16 autocast and feature packs, the original temporal blocks, and FP32 posterior means. Cuts `.5` and `.8` identify `downsamples[5]` and `downsamples[8]`; A, B, and C advance cache slots 0–9, 10–14, and 15–23. The API accepts start/length pairs on one transformed-frame grid, with lengths $4m + 1$. The reference plan uses starts $a = 40 + 20g + p$ for $g, p \in \{0, 1, 2, 3\}$, length 129, origin 40, and 194 source frames for all methods. One numerical detail is essential to bitwise equality: the final $1\times1\times1$ posterior projection must execute *inside* the autocast region. Promoting it to FP32—the apparently safer choice—perturbs the latent by up to 0.028, roughly 0.9% of the latent scale.

Coverage and execution. Host dictionaries map original indices to packed slots. Reads check coverage and restore contiguous native blocks; construction verifies that every selected slot was written, and an adjacent A pair must occupy consecutive packed slots. Compatibility depends on frozen weights, the transformed-frame grid, spatial preprocessing, phase, and source origin. Packs preserve native BF16 values without quantization; empty sets allocate no tensor. Missing positions require a new pack or independent encoding; the planned read path reports an error.

Rolling-bound derivation. After serving block i in both B phases, the stream evicts chunks ending at or before A index $2i + 1$; the next canonical read starts at $2i + 1$ and its shifted counterpart needs one further position, so one retained block and one lookahead block cover both reads. Once copied into the pack, a selected position does not keep its originating rolling chunk alive. At most two canonical A blocks wait in the rolling window, each with at most two positions; a contiguous read adds at most two positions and the current bridge output one B position. Deduplicating aliases gives Equation 9. Shifted-boundary initialization holds only current A/B outputs besides the packs and satisfies the same bound.

Table 3: Reuse boundaries under the original temporal block schedule (one initial frame, then four-frame blocks). Repair extent is the logical prefix before reuse at that cut.

Cut	R	Stride	First read	Repair frames
A (`.5`)	21	2	11	21
B (`.8`)	41	4	10	37

Algorithm 1: Selective materialization and clip execution

Require: Compatible source X with origin o; ordered plan Q; mode μ; frozen segments A, B, C and posterior map P
Ensure: Final latent objects $\mathcal{O}(q)$ for all $q \in Q$

$(K^A, K^B) \leftarrow$ ReadSets(Q, o, μ)
CheckCoverage(X, Q, K^A, K^B)
$(\mathcal{A}, \mathcal{B}) \leftarrow$ BuildPacks(X, K^A, K^B)
for each $q = (a, L)$ in Q **do**
  $d \leftarrow a - o$; $m \leftarrow (L-1)/4$
  $(S_A, S_B, S_C) \leftarrow$ InitialRequestStates
  for $i = 0, \ldots, m$ **do**
    if $R_B(\mu, d, i)$ **then**
      $b \leftarrow$ Read$(\mathcal{B}_{d \bmod 4}, \lfloor d/4 \rfloor + i)$
    else
      if $R_A(\mu, d, i)$ **then**
        $h \leftarrow$ Read$(\mathcal{A}_{d \bmod 2}, \lfloor d/2 \rfloor + U_i)$
      else
        $(h, S_A) \leftarrow A(X[a + I_i], S_A)$
      end if
      $(b, S_B) \leftarrow B(h, S_B)$
    end if
    $(y_i, S_C) \leftarrow C(b, S_C)$
  end for
  $z \leftarrow P(\mathrm{concat}(y_0, \ldots, y_m))$
  $\mathcal{O}(q) \leftarrow$ WriteAndSync(q, z)
  ReleaseRequestStates(S_A, S_B, S_C)
end for
return $\mathcal{O}$

Compute accounting. Including source builds and requests, the reference plan uses 166 ordinary A calls for fixed A and ClipStore versus 214 for fixed B; fixed A executes 512 ordinary B calls, while both deeper policies execute 188 source and 108 request-boundary B calls, saving $\Delta N_B = 512 - (188 + 108) = 216$. Each policy also performs 16 initial A and 16 initial B calls, counted separately because their shapes differ. For a general plan, a nonempty B phase needs $1 + \max K_r^B$ source blocks; fixed A runs B $m_q{+}1$ times per request, ClipStore zero times for a B-origin request and $\min(10, m_q{+}1)$ otherwise.

Output and measurement. Intermediate packs remain GPU resident. Final FP32 latents use an identical four-device layout and `fsync`, with no overlap between writes and computation; the reference group writes 211,943,424 padded bytes, identical across policies. We record retained native payload, unique backing storage at instrumented materialization points, and CUDA allocated peaks during source construction and the complete group.

C Platform, inputs, and measurement detail

Experiments run in a KVM virtual machine with NVIDIA A100-PCIE-40GB GPUs, PyTorch 2.5.1, CUDA 12.4. Independent workers run on four GPUs; every paired method comparison for one video runs on the same GPU. Final files use four approximately 3.7-TiB virtual block devices exposed through Virtio SCSI; we report the exposed configuration without assuming a physical drive type. The inputs are four public samples: OpenCV's surveillance sample `vtest`, a Pexels sunset timelapse from PyAV, the Sintel trailer from GStreamer's collection, and the Big Buck Bunny trailer hosted by W3C. We decode the first 514 real frames of each with PyAV 16.0.1, resize to 480×832, and preserve the RGB arrays; the artifact records URLs, hashes, native dimensions, frame rates, and

exact preprocessing. Method order rotates across videos, plans, and repetitions; paired speedups compare the same video, repetition, and GPU. Technical repetitions and video samples are reported as distinct sources of variation. The model and decoded pixels are resident before the primary timer; a separate experiment starts from compressed video and additionally charges decoding and initial RGB transfer. Runs record source revision, script hashes, request plans, method order, individual timings, feature counts, output bytes, and exactness checks; independent-reference comparisons run outside the timed interval.

D ADDITIONAL RESULTS

Request-level breakdown. Figure 6 shows request compute time by start in the reference plan. B-origin requests read their phase from the first block; the remaining requests compute A/B blocks 0–5, read A during blocks 6–9, and read B from block 10, whereas Fixed B recomputes A throughout blocks 0–9—a difference of 48 ordinary A calls over the plan. Figure 7 separates retained payload, the explicit construction-feature peak, and the whole-group CUDA peak; the build-only CUDA peaks are 11.246 GB (Fixed A) and 11.412 GB (ClipStore).

Overlap. Figure 8 shows the spacing sweep of Section 5: sixteen 129-frame requests at relative starts $8 + s\lfloor i/4 \rfloor + (i \bmod 4)$ with s from 4 to 40. Dense overlap favors selective depth; more dispersed starts require a longer continuously computed source prefix even when only a sparse subset of its features is retained, which explains both storage growth and the declining time advantage.

Temporal grouping. Figure 9 holds the output plan at thirty-two 129-frame requests and partitions consecutive starts into groups of 4, 8, 16, or 32; each group builds its own source and releases its pack after serving. With sixteen requests per group ClipStore takes 57.42 seconds against 63.62 (Fixed A) and 68.51 (Fixed B); with one thirty-two-request group, 51.01 against 58.28 and 63.94. The four-request groups consist entirely of B-origin matches, where ClipStore and Fixed B execute the same path. Panel (b) compares best measured configurations under a retained payload budget, letting the fixed family choose both its cut and group size; these are measured envelopes, not predictions from an automatic selector.

From compressed video. Figure 10 charges decoding, resizing, RGB normalization, and initial GPU transfer before construction. Each policy runs once per video, re-decoding the first 514 frames; the model and kernels are warm and model loading is excluded. Freshly decoded pixels match the canonical arrays, and all 192 final outputs match independent encoding.

Remarks on budgets and placement. The construction plan is the natural optimization unit: grouping requests by source exposes shared intermediate work, and the read-union planner turns a plan into storage and compute predictions before any encoding runs (Equations 6 and 7). The prototype keeps intermediate packs on the GPU and writes final latent objects to storage; moving packs to host memory or disk adds transfers that belong in the same complete-construction objective, with compatibility determining what can be shared and placement determining its cost.

E DECODER-SIDE MEASUREMENTS

All decoder runs use the same pinned model, BF16 autocast, and videos as the encoder evaluation, with latents obtained by encoding 241 frames (61 latent positions).

Determinism and boundary. The official decode path is bitwise repeatable in-process on both videos. Independently decoding a cropped suffix $z_{k:}$ and comparing each latent group's four output frames against the full decode: the difference becomes exactly zero from latent offset 38 (152 output frames) in all four cases (two videos $\times$ crop indices $k \in \{4, 9\}$). Because the decode driver feeds one latent per chunk, any crop index is grid-aligned: the phase condition is trivially satisfied and support is the only obstacle.

Field. Composing Equation 2 through the decoder, with each stage counted at the temporal rate at which it actually runs, gives: input convolution 2, middle blocks 8, first residual group 12, and

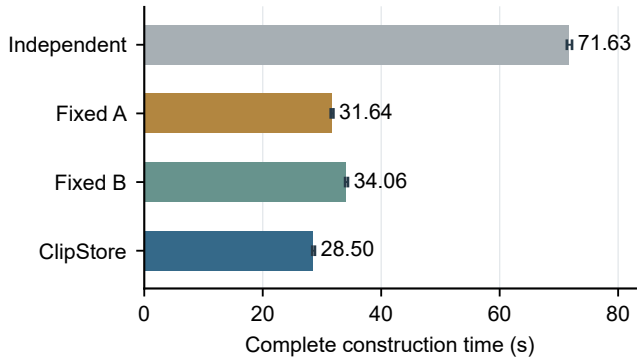

Figure 5: Complete construction of sixteen 129-frame latents. Bars are means of twelve observations per method; whiskers span the observed range. All feature construction and final writes are included.

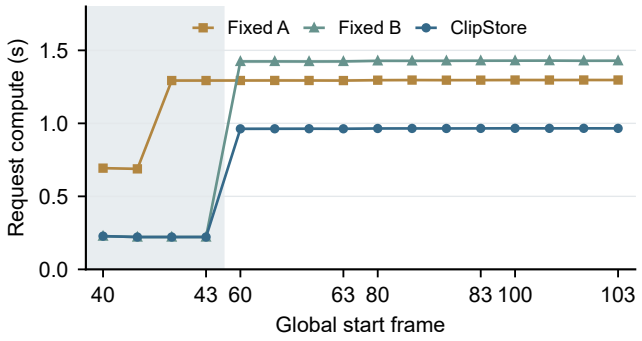

Figure 6: Request compute time by start in the reference plan. Lines are means of twelve observations and bands are full ranges. Shading marks B-origin matches. Source construction and final writes are charged in the complete time, not here.

the first temporal upsample 2, all at latent rate; the second residual group 6 and second temporal upsample 1 at doubled rate; and the third and fourth groups 3 each plus the output convolution 0.5 at quadrupled rate—a 37.5-position field and a safe horizon of 38. A stage-wise forward-influence measurement (hooks at every stage boundary, one latent perturbed) matches this chain exactly: measured reach 2, 10, 22, 24, 30, 31, 34, and 37.3 positions at the successive boundaries. Perturbing a single latent changes exactly the 38 subsequent groups, identically for amplitudes 0.01 and 10.0 and for two perturbed indices; the derived and measured fields agree from both directions. An earlier draft assigned two residual groups to the wrong rate and obtained 46.5; the stage-wise measurement localizes and corrects that assignment.

State continuation. A chunked driver that mirrors the original decode loop reproduces it bitwise. Snapshotting the per-slot causal caches after processing latents $z_{0:k}$ (exclusive endpoint k) and resuming on $z_{k:}$ reproduces the full decode's remaining frames bitwise in six of six cases ($k \in \{10, 20, 40\}$, two videos; worst absolute error 0.0), with zero boundary repair. The snapshot occupies 3.73 GB at 480×832, dominated by the high-resolution slots. Against re-decoding from the safe horizon $\max(0, k - 38)$—the tested alternative—resumption is $1.19\times$ (k=10), $1.47\times$ (k=20), and $2.85\times$ (k=40 of 61) faster; its cost equals a plain crop decode of the suffix while being exact. (For $k \leq 38$ the horizon truncates at zero, so these baselines decode the full sequence; at k=40 the baseline drops two latents.) The comparison quantifies the contract distinction of Section 6: continuation semantics is served bitwise by retained state at zero repair, while independent-clip semantics pays the repair horizon.

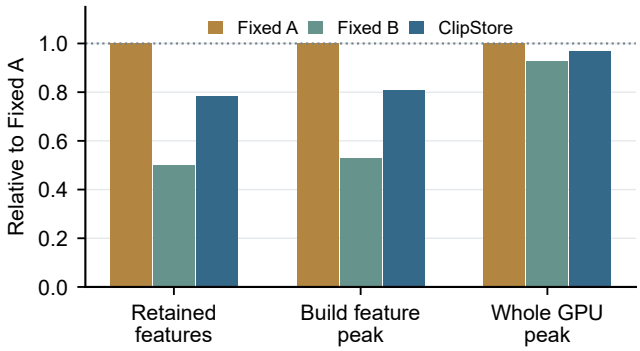

Figure 7: Memory in the reference plan, normalized to Fixed A. Retained payload, explicit construction-feature peak, and whole-group CUDA peak measure different objects; Table 1 gives absolute values.

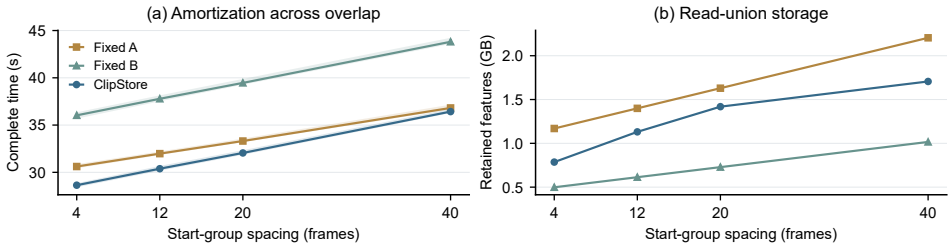

Figure 8: Overlap at fixed request count and length. Increasing the spacing between groups of four phase-distinct starts increases source extent and changes the packed read unions. Means and full ranges across four videos.

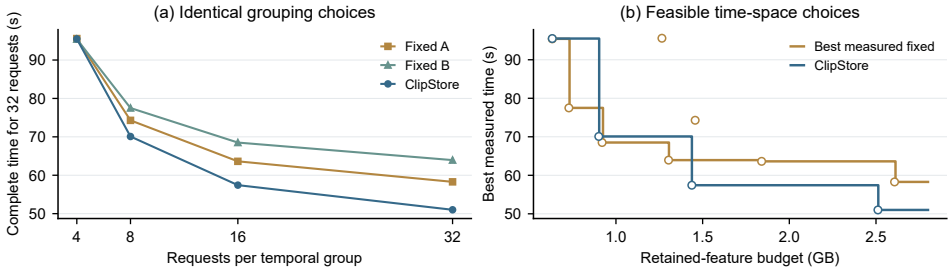

Figure 9: Temporal grouping for the same thirty-two-request output plan. (a) Complete time under identical partitions; means of two repetitions with full-range bands. (b) Best measured time at each retained-feature budget. Circles show measured configurations, including dominated choices.

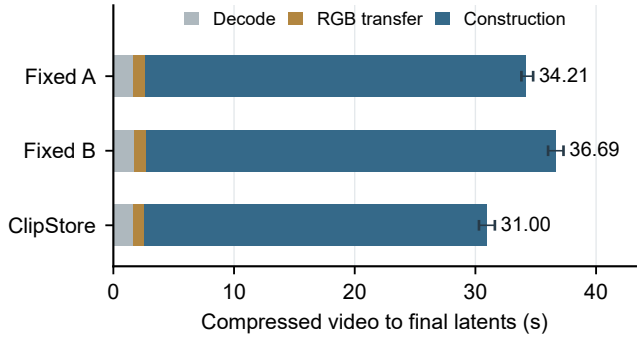

Figure 10: Compressed input to final latent files. Bars show mean decode, initial RGB transfer, and construction time over four videos; whiskers span the observed total-time range. Construction includes final writes and `fsync`.