RETHINKING VIDEO STORAGE FOR AI TRAINING: BREAKING CROSS-FRAME DEPENDENCIES FOR EFFICIENT RANDOM ACCESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large-scale AI training increasingly relies on massive video datasets with random temporal sampling patterns. However, existing video storage paradigms are optimized for sequential playback and employ sophisticated inter-frame reference structures to maximize compression efficiency. When applied to the random-sampling workloads of AI training, these paradigms face substantial decoding-efficiency bottlenecks. Workarounds such as independent JPEG frames and short groups of pictures (GOPs) reduce decoding work at the cost of larger storage and transfer volumes. In this paper, we identify two sources of decode amplification: sequential reconstruction of unrelated frames and reconstruction of the reference chains required by sampled targets. We address both sources by co-designing the decoding schedule and video reference structure. Dependency-aware sparse decoding reconstructs only requested frames and their required references, skipping unrelated reconstruction without transcoding the source videos. Optional offline Dependency-Decoupled Random Access (DDRA) encoding further reduces reference reconstruction by restricting each interior frame to two bounding intra-coded anchors. Together, sparse decoding and DDRA retain inter-frame prediction while bounding reconstruction to at most three frames per requested target, independent of the intra period. Experiments on a large-scale robotics corpus show that, at comparable reconstruction quality, our method achieves $1.45\times$ the loading throughput of LeRobot v3 with 58.7% less media payload.

1 INTRODUCTION

Video-based learning systems are increasingly used in autonomous driving, robotics, and multimodal understanding. Instead of sequential playback, the training workloads repeatedly sample short windows or sparse frames from long recordings, shuffle requests across videos, and revisit the corpus over many epochs. An effective storage path must therefore combine compact representation with fine-grained temporal access and sustained data delivery to accelerators; sequential throughput alone is insufficient.

Conventional video storage exposes a compression–random-access tension under these workloads. General-purpose video codecs exploit inter-frame redundancy to reduce storage and transfer volume Wiegand et al. (2003); Sullivan et al. (2012), but their playback-oriented designs introduce two sources of *decode amplification*. First, container-level seeking locates a synchronization point but does not make decoding frame-selective: conventional pipelines decode every intervening frame sequentially until the target is produced, even though many of these frames are not requested by the training workload. Second, hierarchical reference structures allow B frames to depend on other B frames, so producing one target may additionally require reconstructing a transitive chain of reference frames. Together, redundant sequential decoding and mandatory reference reconstruction consume CPU time without directly producing requested training samples.

Materializing every frame as an independently compressed image avoids temporal decode dependencies, and row- or shard-based formats can then expose individual images to a sampler Leclerc et al. (2023); Pace et al. (2025). A more storage-efficient compromise uses mini-GOP video coding to shorten reference dependency chains, as in our LeRobot Cadene et al. (2026) baseline. Both rep-

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.

resentations better match sparse access, but they sacrifice inter-frame redundancy—either entirely or through frequent intra-coded anchors—and thereby increase persistent-storage use and network-bandwidth demand during training.

We address this tension by co-designing the online decoding schedule and the offline video representation around training requests. First, dependency-aware sparse decoding reconstructs only the requested frames and their direct and transitive references in dependency-respecting order, skipping unrelated frames that a sequential decoder would otherwise process. This removes redundant decoding, but leaves the required reference chains intact. To shorten these chains, we introduce Dependency-Decoupled Random Access (DDRA), an offline encoding configuration that removes B-to-B reference edges and restricts each interior B frame to the two intra-coded anchors that bound its intra period. The resulting dependency closure contains only the target and its two anchors. Together, sparse decoding and DDRA reduce both sources of decode amplification while retaining inter-frame prediction, standards-compliant bitstreams, and the decoded-frame interface expected by existing training pipelines.

We evaluate two settings on three robotics datasets comprising 3,000 episodes and 8,908 camera streams (Table 1). Without offline transcoding, our sparse loader is $1.81\times$, $1.34\times$, and $1.25\times$ faster than PyAV, Decord, and TorchCodec, respectively, based on average loading time across four access patterns. With offline transcoding at comparable reconstruction quality, our H.265 variant achieves $1.45\times$ the throughput of LeRobot v3 and $2.02\times$ that of JPEG, while reducing media payload by 58.7% and 90.0%, respectively. Figure 1 summarizes this storage–throughput trade-off using the same average loading times.

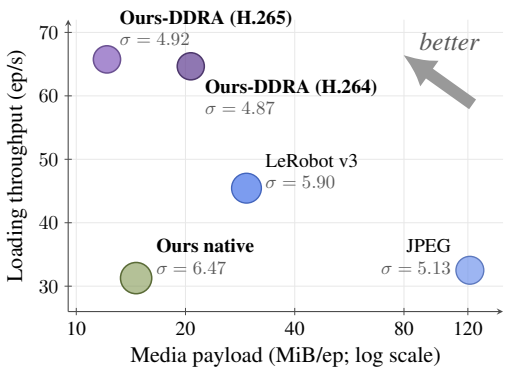


Figure 1: Storage–throughput trade-off for different storage methods. Bubble area encodes σ, the ratio of p95 to mean loading time.

Our contributions are threefold:

- We analyze the video-storage requirements of AI training and identify two sources of decode amplification: sequential decoding of unrelated frames and transitive frame dependencies.
- We combine dependency-aware sparse decoding with DDRA encoding, eliminating unrelated frame reconstruction and bounding each interior target's dependency closure while retaining inter-frame prediction.
- Experiments across three datasets and four access patterns demonstrate faster native video loading and, after quality-aligned transcoding, higher throughput with less media payload than LeRobot v3 and JPEG.

2 Background and Motivation

2.1 Video Coding Framework

General video coding frameworks combine carefully designed modules to exploit different forms of redundancy. Spatial prediction and transform coding reduce correlations within a frame; quantization controls reconstruction precision, and entropy coding exploits statistical regularities in the coded symbols. Temporal redundancy is typically dominant because neighboring frames share much of their visual content (Ma et al., 2020). Inter-frame prediction exploits this overlap through motion compensation from reconstructed reference pictures, so only motion vectors and residual differences need to be coded (Wiegand et al., 2003; Sullivan et al., 2012).

This temporal compression introduces dependencies between pictures. I pictures use no inter-frame prediction and can be reconstructed independently of other pictures. In conventional prediction structures, P pictures depend on earlier reference pictures, whereas B pictures may use both earlier and later references in presentation order. Hierarchical prediction further organizes references across temporal layers (Schwarz et al., 2006; Sullivan et al., 2012; Han et al., 2021). For a simplified view, consider two boundary frames with indices $l < r$ that form temporal layer 0, $\mathcal{L}_0 = \{l, r\}$. Each

subsequent layer inserts the midpoint between consecutive indices from all earlier layers:

$$\mathcal{L}_j = \left\{ \left\lfloor \frac{a+b}{2} \right\rfloor \;\middle|\; a < b \text{ consecutive in } \bigcup_{i=0}^{j-1} \mathcal{L}_i,\ b-a>1 \right\}, \qquad j \geq 1. \tag{1}$$

Frames in higher temporal layers can reference frames in lower layers. These references form chains across temporal layers, so a sparse request can require reconstructing several intermediate pictures. References must precede their dependents in coding order, which can therefore differ from presentation order. Let Q be a nonempty set of requested frames and $\mathrm{Ref}(f)$ the picture-level references of frame f. The pictures required under this graph form the dependency closure

$$D(f) = \{f\} \cup \bigcup_{r \in \mathrm{Ref}(f)} D(r), \qquad D(Q) = \bigcup_{f \in Q} D(f). \tag{2}$$

For a valid schedule π with no previously reconstructed frames, let $R_\pi(Q) \supseteq D(Q)$ contain each reconstructed picture once. The decode amplification is

$$A^{\pi}_{\mathrm{dec}}(Q) = \frac{|R_\pi(Q)|}{|Q|} = 1 + \underbrace{\frac{|R_\pi(Q) \setminus D(Q)|}{|Q|}}_{\text{unrelated reconstruction}} + \underbrace{\frac{|D(Q) \setminus Q|}{|Q|}}_{\text{reference reconstruction}}. \tag{3}$$

Conventional video decoders traverse coding order sequentially and reorder the reconstructed pictures for output. A seek-and-decode path reconstructs a coded prefix from a random-access point, potentially including pictures outside $D(Q)$. Equation 3 separates this *unrelated reconstruction* from *reference reconstruction* within $D(Q) \setminus Q$. Selective decoding can remove the former, but cannot shorten the encoded dependency chains.

Existing video loaders, including PyAV[1], TorchCodec[2], and Decord[3], improve segment-level seeking and decoding parallelism while reducing implementation overhead. However, sequential decoding within accessed segments can still incur unrelated reconstruction, as illustrated by Sequential in Figure 3, while the encoded reference graph remains unchanged. Our dependency-aware sparse decoding eliminates reconstruction outside $D(Q)$ while preserving decoding correctness. DDRA further reduces the remaining reference reconstruction by simplifying the reference structure.

2.2 Existing Data Organizations and Pipelines

Training pipelines locate sampled observations and fetch the corresponding visual data, which is decoded and augmented before being passed to the GPU as RGB tensors (Figure 2, upper left). Dataset formats support this process through different data organizations. For example, WebDataset packs objects into shards for sequential access (Aizman et al., 2019). Parquet stores tabular records, whereas Lance supports random access to multimodal data (Pace et al., 2025).

Data organization determines how samples are located and accessed, while visual encoding influences the cost of transferring and decoding them. During training, such data must first be transferred from storage to compute nodes and then decoded into RGB tensors. In this context, both I/O and decoding can become bottlenecks.

These two costs lead to a compression–access trade-off. Inter-frame coding reduces transfer volume by exploiting temporal redundancy, but accessing a sampled frame may require reconstructing additional reference frames. Independently coded frames remove these dependencies but require a larger payload (Figure 2, lower right). Short groups of pictures (GOPs) provide a compromise, retaining temporal compression while shortening reference chains. For example, our LeRobot v3 (Cadene et al., 2026) baseline uses AV1 with a two-frame GOP. Our approach improves this trade-off by co-designing the encoded reference structure and the decoding schedule for sparse training requests.

Related systems also reduce loading overhead through more efficient data delivery, storage, and reuse. FFCV streamlines loading and preprocessing (Leclerc et al., 2023), while Progressive Compressed Records allow training to use different image fidelities (Kuchnik et al., 2021). Caching

[1] https://github.com/PyAV-Org/PyAV/tree/v18.1.0

[2] https://github.com/meta-pytorch/torchcodec/tree/v0.7.0

[3] https://github.com/dmlc/decord/tree/v0.6.0

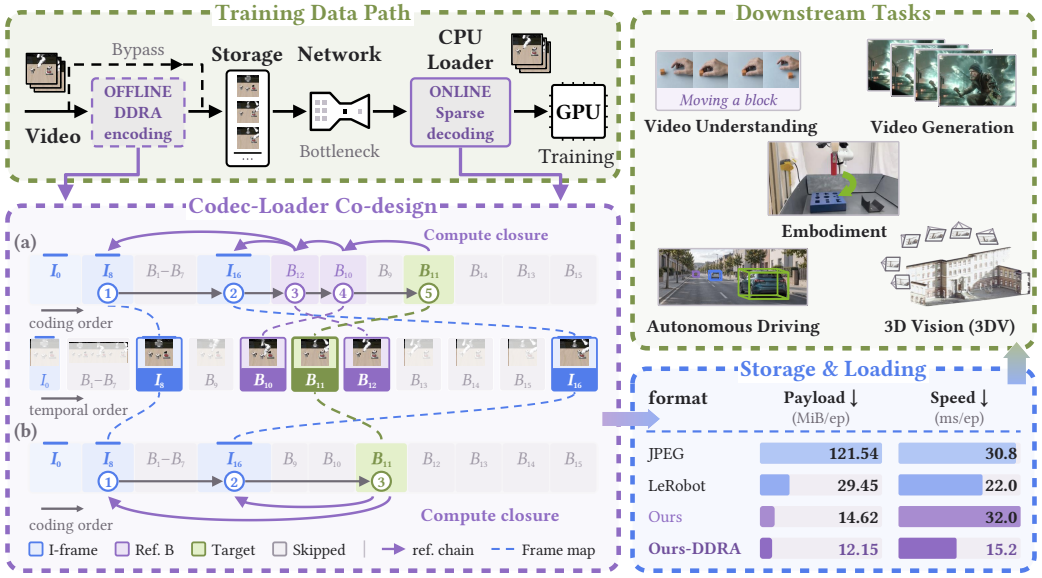


Figure 2: Codec–loader co-design for video-based training. Optional offline DDRA encoding and online dependency-aware sparse decoding deliver requested RGB frames. (a) Hierarchical coding requires intermediate B-frame references; (b) DDRA coding restricts each interior frame to two intra-coded anchors, reducing the illustrated dependency closure.

systems such as CoorDL, Quiver, SHADE, and Seneca reduce repeated data access (Mohan et al., 2021; Kumar & Sivathanu, 2020; Khan et al., 2023; Desai et al., 2026), and SAND enables reuse of preprocessing outputs for video training (Ye et al., 2025). Video storage systems also optimize how the data is represented and accessed: VStore selects formats for analytics, VSS manages layouts and caches, and TASM uses tiling for spatial access (Xu et al., 2019; Haynes et al., 2021; Daum et al., 2021). Our approach complements these efforts by reducing the cost for transferring and decoding when no previous results are unavailable for reuse.

3 Method

Our approach co-designs the decoding schedule and the encoding reference graph to reduce the two components of decode amplification in Equation 3. Online dependency-aware sparse decoding removes most of the unrelated reconstruction. Optional offline DDRA encoding further reduces the reference frames by restricting the reference frames of each B-frame to two intra-coded anchors. Figure 2 shows how the two components fit into the training data path.

3.1 Dependency-Aware Sparse Decoding

Video containers typically store coded frames as individually addressable samples, making selective frame access possible. To respect the reference dependencies in Section 2.1, selective reconstruction must include the closure $D(Q)$ defined in Equation 2.

Given requested indices $\boldsymbol{q}$ with distinct-frame set Q, we first perform a header-parsing pass to construct the reference graph. We select a segment $\mathcal{S}$ from a preceding random-access anchor through the last target in decoding order, extending it backward if required references are unavailable (Appendix A.4). For each frame f, we define $\mathrm{Ref}(f)$ as the union of the active reference lists across its slice headers. A breadth-first traversal from Q then yields the dependency closure $D(Q)$ (Equation 2). We then traverse $\mathcal{S}$ in decoding order, reconstructing each frame in $D(Q)$ once and skipping pixel reconstruction for the remaining frames (Figure 2(a)). This order ensures that references are reconstructed before their dependent pictures. Only frames in Q are converted to RGB, and the outputs are returned in the original request order, including duplicates (Algorithm 1).

Algorithm 1 Dependency-aware sparse decoding with a validated reference graph

Input: Encoded video V, decoding order π, requested indices $\boldsymbol{q}$.
Output: Requested RGB frames in the original request order.

$Q \leftarrow \mathrm{set}(\boldsymbol{q})$
$(\mathcal{S}, \mathrm{Ref}) \leftarrow \mathrm{SelectAndParse}(V, Q)$
$D(Q) \leftarrow \mathrm{BFSClosure}(\mathrm{Ref}, Q)$
$\sigma \leftarrow \mathrm{InitState}(V, \mathcal{S}); \quad X \leftarrow \varnothing$
for $f \in \mathcal{S}$ in order π **do**
  if $f \in D(Q)$ **then**
    $(x, \sigma) \leftarrow \mathrm{DecodeFrame}(V[f], \sigma)$
    if $f \in Q$ **then** $X[f] \leftarrow \mathrm{RGB}(x)$
  else
    $\sigma \leftarrow \mathrm{UpdateState}(V[f], \sigma)$
  end if
end for
return $[X[f] : f \in \boldsymbol{q}]$

Skipping reconstruction still requires preserving the decoder state σ. For each skipped frame, a state-only path updates picture-order counts and reference management in the decoded picture buffer (DPB) to maintain consistency with the encoder's DPB state. On a validated reference graph, the schedule thus satisfies $R_\pi(Q) = D(Q)$ and removes unrelated reconstruction. The remaining decoding cost is determined by the frame-reference structure, motivating DDRA.

3.2 Dependency-Decoupled Encoding

Dependency-Decoupled Random Access (DDRA) removes the interior reference chains described in Section 2.1 (Equation 1). Let T_I denote the intra period and $k \geq 0$ the interval index. Each interior frame B_t references only the bounding anchors I_{kT_I} and $I_{(k+1)T_I}$. In presentation order, the resulting sequence and reference sets are

$$\begin{aligned} &I_0, \ldots, I_{kT_I},\ B_{kT_I+1}, \ldots, B_{(k+1)T_I-1},\ I_{(k+1)T_I}, \ldots, \\ &\mathrm{Ref}(I_t) = \varnothing,\ \mathrm{Ref}(B_t) = \{I_{kT_I}, I_{(k+1)T_I}\}, \qquad kT_I < t < (k+1)T_I. \end{aligned} \tag{4}$$

The DDRA makes reference depth independent of temporal distance. Every interior frame is one prediction step from independently coded anchors; increasing T_I or changing the target's position within the interval does not add intermediate references. Sparse decoding can therefore skip reconstruction of all unrequested interior frames, adding at most two anchors per target (Figure 2(b)). Thus, for any nonempty request Q,

$$|D_{\mathrm{DDRA}}(Q)| \leq 3|Q|, \qquad A_{\mathrm{dec}}^{\mathrm{DDRA}}(Q) = \frac{|D_{\mathrm{DDRA}}(Q)|}{|Q|} \leq 3. \tag{5}$$

The bound is attained by a single interior target. For requests containing multiple frames, anchor sharing can reduce the actual amplification; Section 3.3 analyzes this dependence on the request pattern. The intra period controls anchor frequency without changing this amplification bound. Longer intervals require fewer intra-coded anchors, but more distant predictors can increase the bits needed to encode interior frames. Removing B-to-B references also limits prediction flexibility. DDRA therefore trades compression efficiency for bounded reference reconstruction while retaining temporal prediction. We evaluate this storage–access trade-off in Section 4.

3.3 Decode-Amplification Analysis

For batched requests, DDRA's amplification depends on the number of distinct additional anchors. Let $K = |Q|$, and let $\mathcal{A}(Q)$ contain the bounding anchors of every interval with a requested interior frame. Then

$$D_{\mathrm{DDRA}}(Q) = Q \cup \mathcal{A}(Q), \qquad A_{\mathrm{dec}}^{\mathrm{DDRA}}(Q) = 1 + \frac{|\mathcal{A}(Q) \setminus Q|}{K}. \tag{6}$$

For K interior targets in one interval, only $K + 2$ pictures are reconstructed, giving amplification $1 + 2/K$. Across intervals, the cost depends on distinct anchors rather than skipped interior frames.

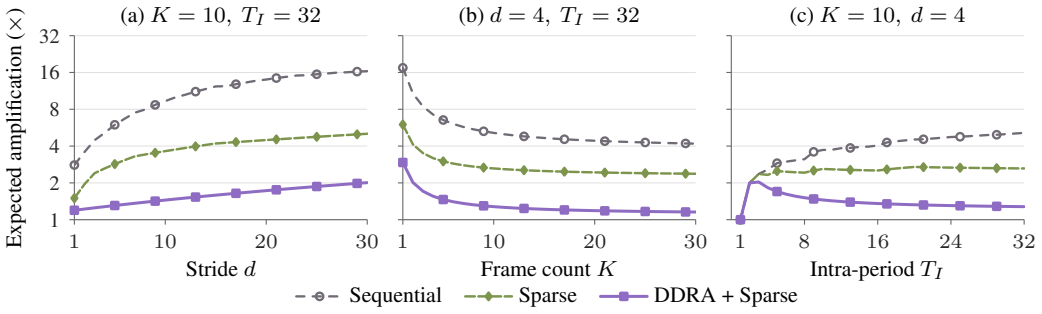


Figure 3: Exact expected decode amplification for Sequential, Sparse, and DDRA + Sparse under the periodic counting model. Sparse denotes dependency-aware sparse decoding (Section 3.1) and DDRA denotes the encoding configuration in Section 3.2.

The request pattern and intra period therefore jointly determine how far amplification falls below the per-frame upper bound.

To isolate the effects of scheduling and reference structure, we compare Sequential, Sparse, and DDRA + Sparse. Sequential and Sparse use the hierarchy in Equation 1 between successive intra anchors. Requests take the form

$$Q(s, K, d) = \{F_{s+jd} \mid 0 \le j < K\}, \qquad s \sim \text{Uniform}\{0, \ldots, T_I - 1\}, \tag{7}$$

where F_t is frame t, d is the stride, and $T_I \ge 1$. With $K = 10$, increasing stride from 1 to 30 raises Sequential's expected amplification from 2.80 to 16.40 (Figure 3(a)). At stride 30, Sparse reduces this value to 5.04, and DDRA further reduces it to 2.01 by shortening the reference closure. At $d = 4$, increasing K from 1 to 30 amortizes shared anchors across more targets, lowering DDRA's expectation from 2.94 to 1.16 (Figure 3(b)). The intra period controls how frequently new anchors are coded. At $K = 10$ and $d = 4$, increasing T_I from 4 to 32 lowers DDRA's expected amplification from 1.83 to 1.28, while Sparse changes from 2.33 to 2.63 (Figure 3(c)). Longer intervals let more targets reuse each anchor pair under DDRA; the hierarchical graph also introduces interior reference reconstruction. These expectations isolate pixel reconstruction; loading time also depends on data transfer and the computational cost per picture.

4 Evaluation

4.1 Experimental Setup

We evaluate data loading performance on a corpus consisting of 3,000 episodes and 8,908 camera streams curated from RealSourceWorld, RoboCasa (Nasiriany et al., 2026), and Table30v2 (Ma et al., 2026). All loading experiments store every representation as episode-level Lance records on OSS (object storage) and read it through this common storage path. PyAV, Decord, TorchCodec, and Ours native consume the original bitstreams without transcoding; Ours native uses dependency-aware sparse decoding. The transcoded baselines are LeRobot v3 (Cadene et al., 2026) and JPEG. LeRobot v3 does not mandate a particular video codec; we use its recommended configuration[4] for this comparison. Ours-DDRA (H.264/H.265) uses bitstreams with open GOPs and intra period $T_I = 32$. We ensure the selected representations' average PSNR differs from the LeRobot reference by at most 0.18 dB for fairness (Appendix A.2).

We first validate sparse decoding correctness through pixelwise comparison with full sequential decoding (Appendix A.5). Our evaluation then examines efficiency, training performance, and operating limits. We compare loading time and storage cost across methods (Section 4.2). We then evaluate how training-data compression affects VLA, WAM, and VLM task performance (Section 4.3). We also vary temporal sampling and loader resources to assess the robustness of loading speedup, constrain OSS bandwidth to identify transfer-limited regimes, and sweep the intra period to characterize

[4] https://huggingface.co/docs/lerobot/video_encoding_parameters

Table 1: Batch-loading efficiency on three datasets. Average and p95 loading times are macro-averaged across datasets. For methods requiring offline transcoding, reconstruction distortion is aligned to LeRobot v3 using RGB PSNR. **bold** marks the best value within each method group.

		No offline transcoding				**Offline transcoding required**			
Pattern	Stat.	PyAV	Decord	Torch Codec	**Ours** native	JPEG	LeRobot v3	**Ours-DDRA** H.264	**Ours-DDRA** H.265
Loading time (ms/batch) ↓									
Contiguous	Average	336.5	305.2	278.2	**226.7**	251.0	170.2	123.6	**118.2**
	p95	2286.0	1916.3	1963.7	**1348.3**	1274.6	982.2	573.2	**521.5**
Random	Average	594.2	407.1	383.3	**297.8**	248.8	185.1	**134.1**	140.6
	p95	2895.7	2088.3	1952.8	**1921.3**	1238.9	1218.2	**633.4**	714.9
Stride-2	Average	438.5	314.6	294.9	**238.5**	242.1	173.0	120.3	**111.4**
	p95	2486.6	1851.0	**1685.1**	1729.0	1198.3	958.9	626.7	**552.5**
Stride-4	Average	485.6	349.3	324.1	**260.2**	242.5	175.8	116.9	**116.5**
	p95	2598.1	1959.5	1760.4	**1621.7**	1336.5	995.1	**575.5**	605.6
Payload (MiB/ep) ↓		**14.62**	**14.62**	**14.62**	**14.62**	121.54	29.45	20.68	**12.15**
Mean CPU cores ↓		13.44	14.08	13.76	**13.41**	14.70	11.68	11.67	**11.46**
PSNR (dB) ↑		—	—	—	—	41.393	41.473	41.294	**41.496**

the loading-time–payload trade-off (Section 4.4). Detailed experimental protocols are provided in Appendices A–D.

4.2 Loading Efficiency and Storage Cost

Loading results. All methods use one 16-CPU, 64-GiB node, 32 workers, and batch size eight. The timed path covers OSS reads, RGB decoding, and metadata return (Appendix A.3). Sparse decoding improves average loading time in all four patterns without changing the stored video (Table 1). Averaged across patterns, Ours native is $1.81\times$, $1.34\times$, and $1.25\times$ faster than PyAV, Decord, and TorchCodec, respectively. It also has the lowest native p95 in all patterns except for Stride-2. DDRA transcoding further improves loading efficiency. In random access mode, the H.264 and H.265 DDRA representations reduce average loading time from 297.8 to 134.1 and 140.6 ms/batch, respectively ($2.22\times$ and $2.12\times$). Both have lower mean and p95 loading times than LeRobot v3 and JPEG in every main-table pattern.

Payload–throughput trade-off. Among methods requiring offline transcoding, Ours-DDRA reduces media pyload for better storage/transmission efficiency and improves batch loading throughput at aligned quality (Figure 1). Its H.264/H.265 representations provide 1.42–$1.45\times$ LeRobot v3's throughput with 29.8–58.7% less media payload, and 1.99–$2.02\times$ JPEG's throughput with 83.0–90.0% less payload. H.265 requires less storage than H.264 while achieving comparable loading times. We therefore use H.265 for the subsequent training-performance evaluation.

4.3 Training Performance under Compression

We evaluate how DDRA compression affects downstream performance for VLA and WAM on LIBERO (Liu et al., 2023) and VLM on MVBench (Li et al., 2024), with fixed training budgets within each model family. The VLA and VLM experiments use $\pi_{0.5}$ (Physical Intelligence et al., 2025) and Qwen3.5-2B, respectively. Our WAM experiments follow OpenWAM (Wang et al., 2026). Figure 4 compares task performance against the compression factor relative to Original. Appendix B provides the protocols and numerical results.

DDRA (H.265) requires approximately 50% less storage and media-transfer bandwidth than LeRobot v3 while maintaining comparable VLA and WAM performance. Following the procedure in prior work Liang et al. (2026), we compute approximate 95% confidence intervals for the success-rate (SR) differences of LeRobot v3 and DDRA relative to Original. Under this approximation, DDRA (H.265) shows no statistically significant SR change for either VLA or WAM at approxi-

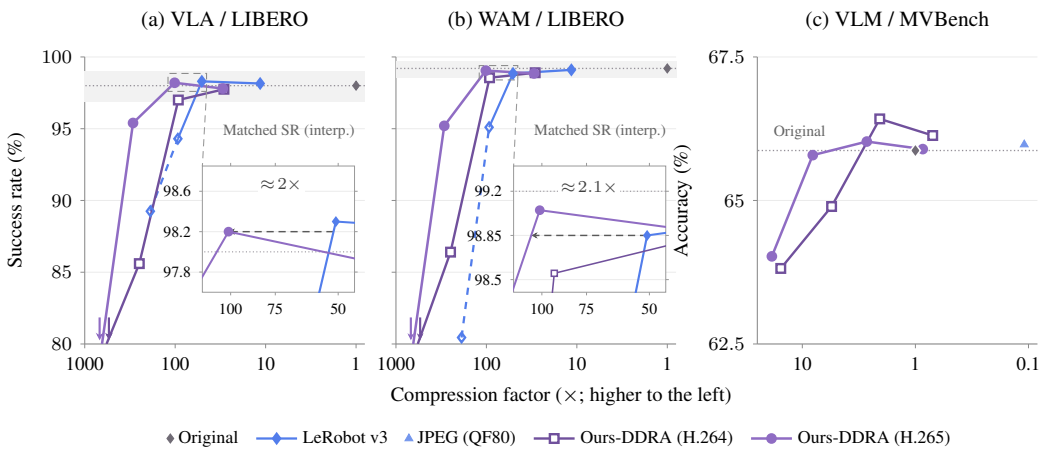


Figure 4: Training performance under compression. DDRA (H.265) approximately halves storage and bandwidth requirements relative to LeRobot v3 at comparable VLA/WAM performance, while high-quality transcoding need not reduce VLM accuracy. Compression factors use the original visual representations as anchors, explaining their different ranges.

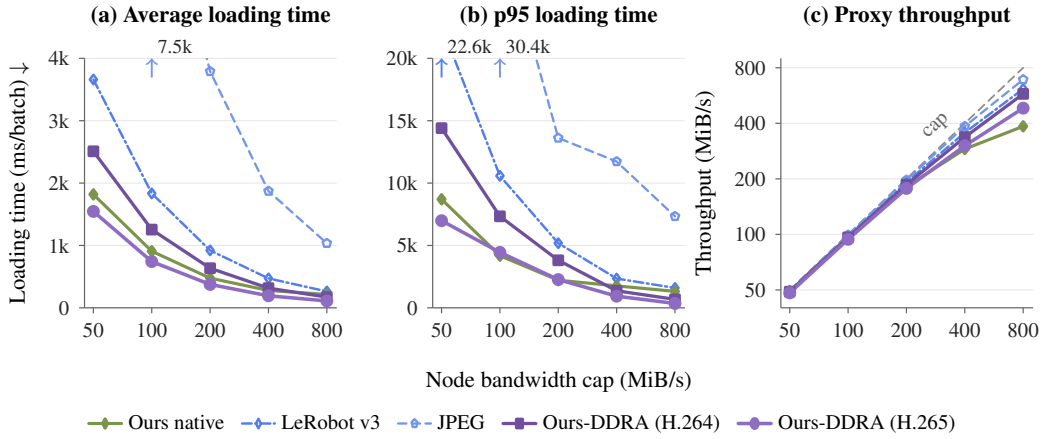


Figure 5: Loading ten contiguous frames per camera under node-wide bandwidth caps of 50–800 MiB/s. (a) Average loading time and (b) p95 loading time over 100 measured batches, both macro-averaged across datasets. (c) Aggregate object-storage response throughput over the full proxy window. The dashed line in (c) marks the cap. Arrows label values beyond the axis limits.

mately 100× compression relative to Original ($p > 0.05$; Appendix B.3). For VLM, transcoding does not necessarily reduce downstream accuracy when video quality remains high. One possible explanation is that moderate compression brings the visual distribution of the training videos closer to that of the test videos.

4.4 Ablation Study

Bandwidth-Constrained Loading. Figure 5 compares Ours native, LeRobot v3, JPEG, and DDRA H.264/H.265 under constrained bandwidth. DDRA sustains higher sampled-episode loading throughput than LeRobot v3 and JPEG in this setting. At a node-wide cap of 100 MiB/s, DDRA H.264 and H.265 achieve 6.38 and 10.79 ep/s, respectively, compared with 4.36 ep/s for LeRobot v3 and 1.06 ep/s for JPEG. At 800 MiB/s, Ours-DDRA (H.264) exceeds Ours native in loading throughput. This crossover is consistent with decoding efficiency having a greater influence as transfer constraints relax. Appendix D details the sampling and node-wide bandwidth protocol.

Figure 6: Influence of intra-period.

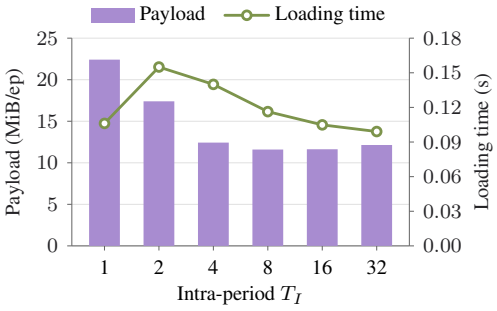


Table 2: Cumulative component ablation.

Pattern	Metric ↓	Sequential	+ Sparse decoding	+ DDRA encoding
Contiguous	Step (s)	0.3380	0.2267	**0.1236**
	Frames/ep	82.1	43.5	**34.8**
Random	Step (s)	0.6683	0.2978	**0.1341**
	Frames/ep	233.9	92.3	**43.1**
Stride-4	Step (s)	0.5057	0.2602	**0.1169**
	Frames/ep	154.6	78.2	**36.9**

Access and Resource Sensitivity. DDRA's loading gains persist across all 28 tested access/resource configurations (selected settings in Figure 7 of Appendix C). Varying requested frame count, stride, random span, batch size, workers, and CPU allocation gives speedups over Ours native of 1.44–4.41× for H.264 and 1.59–3.27× for H.265. The one-variable sweeps use 100 measured batches per point; Appendix C gives their settings and scaling analysis.

Influence of Intra-period. We evaluate the effects of different intra periods on video payload and loading speed. We use DDRA (H.265) encoding and an access pattern with stride $s = 4$; the results are shown in Figure 6. It achieves its smallest payload at $T_I = 8$ and its lowest average loading time at $T_I = 32$. Moving from 8 to 32 lowers average loading time by 14.8%, while increasing payload by 4.7%. This illustrates a trade-off: a larger intra period allows more sampled frames to share the same pair of keyframes, reducing the number of distinct reference frames that must be decoded. However, the greater temporal distance to these anchors can make inter-frame prediction less efficient and increase the total bitrate.

Component Ablation. Table 2 quantifies the incremental gains of sparse decoding and DDRA encoding in the proposed pipeline. All three configurations use H.265 with $T_I = 32$, identical sampled requests, and the same loading resources (Appendix A). Sequential decodes each requested intra period in coding order through its last target, skipping untouched periods and unused tails. The first comparison keeps the encoded videos unchanged and replaces sequential reconstruction with sparse decoding. The second keeps sparse decoding enabled and measures the additional benefit of the DDRA encoding configuration.

Each stage reduces reconstruction work and loading time across all three access patterns. Under random access, sparse decoding reduces reconstructed frames from 233.9 to 92.3 per episode and batch-loading time from 668.3 to 297.8 ms, yielding a 2.24× speedup. Adding DDRA further reduces these values to 43.1 frames per episode and 134.1 ms per batch, yielding an additional 2.22× speedup over sparse decoding alone. The frame counts support the intended reduction in reconstruction work, while the timings quantify the incremental loading benefit of each stage. Across patterns, the combined design achieves a 2.73–4.98× speedup over Sequential.

5 Conclusion

This paper explores workload-aware video storage for AI training. By modifying codec reference structures, DDRA bridges efficient compression and efficient random access. On our controlled three-dataset benchmark, Ours-DDRA (H.264) and (H.265) reduce average loading time in random access mode by 2.22× and 2.12×, respectively, relative to native sparse H.264 loading. Across four access patterns they achieve sampled-episode loading throughputs of 64.66 and 65.75 ep/s while using 83.0–90.0% less media payload than JPEG. These results show that bounded reference dependencies provide a practical operating point between playback-oriented compression and image-oriented random access. The preferred representation depends on request span and bandwidth; the reported loading rates do not establish full-tensor delivery rates or training-time speedups.

AI USE STATEMENT

We used generative AI tools primarily to polish the manuscript and assist with code development and debugging.

REFERENCES

Alex Aizman, Gavin Maltby, and Thomas M. Breuel. High performance I/O for large scale deep learning. In *2019 IEEE International Conference on Big Data (Big Data)*, pp. 5965–5967. IEEE, 2019. doi: 10.1109/BigData47090.2019.9005703. URL https://arxiv.org/abs/2001.01858.

Remi Cadene, Simon Alibert, Francesco Capuano, Michel Aractingi, Adil Zouitine, Pepijn Kooijmans, Jade Choghari, Martino Russi, Caroline Pascal, Steven Palma, Mustafa Shukor, Jess Moss, Alexander Soare, Dana Aubakirova, Quentin Lhoest, Quentin Gallouédec, and Thomas Wolf. LeRobot: An open-source library for end-to-end robot learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=CiZMMAFQR3.

Maureen Daum, Brandon Haynes, Dong He, Amrita Mazumdar, and Magdalena Balazinska. TASM: A tile-based storage manager for video analytics. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 1775–1786. IEEE, 2021. doi: 10.1109/ICDE51399.2021.00156. URL https://db.cs.washington.edu/projects/visualworld/tasm.pdf.

Omkar Desai, Ziyang Jiao, Shuyi Pei, Janki Bhimani, and Bryan S. Kim. Preparation meets opportunity: Enhancing data preprocessing for ML training with seneca. In *24th USENIX Conference on File and Storage Technologies (FAST 26)*, pp. 221–238, Santa Clara, CA, 2026. USENIX Association. ISBN 978-1-939133-53-3. URL https://www.usenix.org/conference/fast26/presentation/desai.

Jingning Han, Bohan Li, Debargha Mukherjee, Ching-Han Chiang, Adrian Grange, Cheng Chen, Hui Su, Sarah Parker, Sai Deng, Urvang Joshi, Yue Chen, Yunqing Wang, Paul Wilkins, Yaowu Xu, and James Bankoski. A technical overview of AV1. *Proceedings of the IEEE*, 109(9):1435–1462, 2021. doi: 10.1109/JPROC.2021.3058584. URL https://doi.org/10.1109/JPROC.2021.3058584.

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*, SIGMOD '21, pp. 685–696. Association for Computing Machinery, 2021. doi: 10.1145/3448016.3459242. URL https://db.cs.washington.edu/projects/visualworld/vss.pdf.

Redwan Ibne Seraj Khan, Ahmad Hossein Yazdani, Yuqi Fu, Arnab K. Paul, Bo Ji, Xun Jian, Yue Cheng, and Ali R. Butt. SHADE: Enable fundamental cacheability for distributed deep learning training. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*, pp. 135–152, Santa Clara, CA, 2023. USENIX Association. ISBN 978-1-939133-32-8. URL https://www.usenix.org/conference/fast23/presentation/khan.

Michael Kuchnik, George Amvrosiadis, and Virginia Smith. Progressive compressed records: Taking a byte out of deep learning data. *Proceedings of the VLDB Endowment*, 14(11):2627–2641, 2021. doi: 10.14778/3476249.3476308. URL https://www.vldb.org/pvldb/vol14/p2627-kuchnik.pdf.

Abhishek Vijaya Kumar and Muthian Sivathanu. Quiver: An informed storage cache for deep learning. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pp. 283–296, Santa Clara, CA, 2020. USENIX Association. ISBN 978-1-939133-12-0. URL https://www.usenix.org/conference/fast20/presentation/kumar.

Guillaume Leclerc, Andrew Ilyas, Logan Engstrom, Sung Min Park, Hadi Salman, and Aleksander Madry. FFCV: Accelerating training by removing data bottlenecks. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12011–12020. IEEE,

2023. doi: 10.1109/CVPR52729.2023.01156. URL https://openaccess.thecvf.com/content/CVPR2023/html/Leclerc_FFCV_Accelerating_Training_by_Removing_Data_Bottlenecks_CVPR_2023_paper.html.

Kunchang Li, Yali Wang, Yinan He, Yizhuo Li, Yi Wang, Yi Liu, Zun Wang, Jilan Xu, Guo Chen, Ping Luo, Limin Wang, and Yu Qiao. MVBench: A comprehensive multi-modal video understanding benchmark. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. URL https://arxiv.org/abs/2311.17005.

Yushen Liang, Yue Peng, Baosheng Jin, Tianluo Zhang, Xinyu Zhang, Shuyi Zhou, Zhuoran Chen, Xinqi Liu, and Shenji Wan. TFP: Temporally conditioned memory-fusion policies for visuomotor learning. *arXiv preprint arXiv:2607.08283*, 2026. URL https://arxiv.org/abs/2607.08283v3. Accepted to the SemRob Workshop at RSS 2026.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023. URL https://arxiv.org/abs/2306.03310.

Siwei Ma, Xinfeng Zhang, Chuanmin Jia, Zhenghui Zhao, Shiqi Wang, and Shanshe Wang. Image and video compression with neural networks: A review. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(6):1683–1698, 2020. doi: 10.1109/TCSVT.2019.2910119. URL https://arxiv.org/abs/1904.03567.

Yunchao Ma, Ze Chen, Erjin Zhou, Ziming Liu, Haowei Zhang, Kai Liu, and Haoqiang Fan. Table30 V2: Evaluating generalized models by real robots at scale. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 4461–4467, June 2026. URL https://openaccess.thecvf.com/content/CVPR2026W/GigaBrainChallenge/html/Ma_Table30_V2_Evaluating_Generalized_Models_by_Real_Robots_at_Scale_CVPRW_2026_paper.html.

Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. Analyzing and mitigating data stalls in DNN training. *Proceedings of the VLDB Endowment*, 14(5):771–784, 2021. doi: 10.14778/3446095.3446100. URL https://www.vldb.org/pvldb/vol14/p771-mohan.pdf.

Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2603.04356.

Weston Pace, Chang She, Lei Xu, Will Jones, Albert Lockett, Jun Wang, and Raunak Shah. Lance: Efficient random access in columnar storage through adaptive structural encodings. *arXiv preprint arXiv:2504.15247*, 2025. URL https://arxiv.org/abs/2504.15247.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025. URL https://arxiv.org/abs/2504.16054.

Heiko Schwarz, Detlev Marpe, and Thomas Wiegand. Analysis of hierarchical B pictures and MCTF. In *2006 IEEE International Conference on Multimedia and Expo*, pp. 1929–1932. IEEE, 2006. doi: 10.1109/ICME.2006.262934. URL https://doi.org/10.1109/ICME.2006.262934.

Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, 2012. doi: 10.1109/TCSVT.2012.2221191. URL https://ieeexplore.ieee.org/document/6316136.

Yuran Wang, Siqiao Huang, Mingleyang Li, Chenhao Zhang, Jiaqi Liang, Weiyang Jin, Yue Chen, Xuemin Chi, Donghao Zhou, Qize Yu, Yu-Kai Wang, Yuhan Rui, Shenzhe Yao, Zhen Yuan, Zhenhao Shen, Kefei Zhu, Zijie Zhu, Ning Gao, Xiaowei Chi, Guanqi He, Shanghang Zhang, Hao Dong, Lin Shao, and Hang Zhao. OpenWAM: An open, modular exploration towards systematic world-action model pretraining. *arXiv preprint arXiv:2609.07398*, 2026. URL https://arxiv.org/abs/2609.07398.

Thomas Wiegand, Gary J. Sullivan, Gisle Bjøntegaard, and Ajay Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003. doi: 10.1109/TCSVT.2003.815165. URL https://ieeexplore.ieee.org/document/1218189.

Tiantu Xu, Luis Materon Botelho, and Felix Xiaozhu Lin. VStore: A data store for analytics on large videos. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, pp. 16:1–16:17. Association for Computing Machinery, 2019. doi: 10.1145/3302424.3303971. URL https://arxiv.org/abs/1810.01794.

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*, SOSP '25, pp. 589–605. Association for Computing Machinery, 2025. doi: 10.1145/3731569.3764847. URL https://ina.kaist.ac.kr/assets/bibliography/SAND.pdf.