PRESTO: Hybrid CPU-GPU Preprocessing Framework for Video-based AI Inference System

Jihyuk Lee
Chung-Ang University
Seoul, Republic of Korea
wlgur0909@cau.ac.kr

Dongsu Han
KAIST
Daejeon, Republic of Korea
dhan.ee@kaist.ac.kr

Jaehong Kim
Carnegie Mellon University
Pittsburgh, USA
jaehong4@andrew.cmu.edu

Abstract

The growing adoption of video-based AI models has created a pressing demand for high throughput, low latency inference systems. However, existing preprocessing frameworks—whether CPU or GPU based—struggle to keep up with the computational burdens of video decoding and data augmentation, resulting in suboptimal GPU utilization and degraded inference system performance.

In this paper, we present PRESTO, a high-performance hybrid CPU-GPU preprocessing framework tailored for video-based AI inference systems. PRESTO integrates a hybrid preprocessing scheduler to dynamically balance CPU and GPU workloads, leverages selective decoding to eliminate unnecessary frame processing, and introduces a custom GPU Memory Manager that enables pipelined preprocessing and efficient GPU memory reuse. Through evaluation on the video captioning task, we show that PRESTO achieves up to 4.37× higher throughput and 2.72× lower latency compared to the de facto baselines, while reducing cloud costs by up to 75%.

CCS Concepts

• **Computer systems organization** → **Cloud computing**; **Heterogeneous (hybrid) systems**.

Keywords

Hybrid CPU-GPU Preprocessing Framework, Video based AI model inference, Cloud Computing

ACM Reference Format:
Jihyuk Lee, Dongsu Han, and Jaehong Kim. 2025. PRESTO: Hybrid CPU-GPU Preprocessing Framework for Video-based AI Inference System. In *3rd International Workshop on Networked AI Systems (NetAISys '25), June 23–27, 2025, Anaheim, CA, USA*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3711875.3736688

1 Introduction

As video-based AI models become increasingly prevalent across domains such as autonomous driving [3], smart surveillance [15], and real-time analytics [8], the demand for inference servers capable of processing video streams in real time has grown rapidly. These trends are further accelerated by advances in multimodal large language models [1, 9], which have achieved outstanding performance across a variety of high-level reasoning tasks [4, 5].

However, deploying such models at scale introduces significant system-level challenges—most notably, the need to decode and preprocess large volumes of video data efficiently. Traditional CPU-only or GPU-only decoding methods often fall short under heavy workloads, limiting overall throughput and responsiveness. To address this, more efficient preprocessing strategies and supporting frameworks are required for high-throughput video-based AI systems.

In this paper, we first conduct a detailed study of widely used video preprocessing frameworks (PyTorch DataLoader [12] and NVIDIA DALI [11]) and identify the fundamental limitations of CPU-only and GPU-only approaches. Based on our findings, we propose PRESTO, which addresses these limitations by utilizing CPU and GPU resources together for preprocessing operations. The proposed framework integrates GPU-accelerated and CPU-based decoders to maximize throughput and minimize latency. Our design carefully considers three critical aspects: efficient resource allocation and scheduling across heterogeneous hardware, optimized video decoding strategies that maximize preprocessing performance, and effective GPU memory management that considers the characteristics of video AI workloads.

Through experimental validation, we demonstrate that the prototype of PRESTO significantly enhances throughput for video-based AI inference systems while keeping latency low. PRESTO improves end-to-end throughput (i.e., video requests per second) of HD-VILA [14], one of the widely adopted video captioning models, by up to 4.36× over popular CPU-based pipelines (PyTorch DataLoader with FFmpeg) and 3.48× over GPU-based solutions (NVIDIA DALI with NVDEC). PRESTO can reduce cloud instance requirements by up to 75%, resulting in cost savings and reduced carbon footprint for large-scale deployments. We believe PRESTO opens up new research directions toward distributed, multi-cluster systems for scalable video-based AI inference.

2 Background & Related Works

Video-based AI model inference system. As clients submit multiple video-inference requests to the server, the videos in those requests are queued and processed sequentially through a *preprocessing pipeline*: it decodes each video into frames, resizes and normalizes every frame, and packs the augmented frames into a single model-input tensor. This pipeline is managed by a preprocessing framework such as PyTorch DataLoader, which typically leverages multiple processors to process videos in parallel. Once a preprocessed input tensor is ready, the batch is passed to the model

MobiSys '25, Anaheim, CA, USA

ACM ISBN 979-8-4007-1453-5/25/06
https://doi.org/10.1145/3711875.3736688

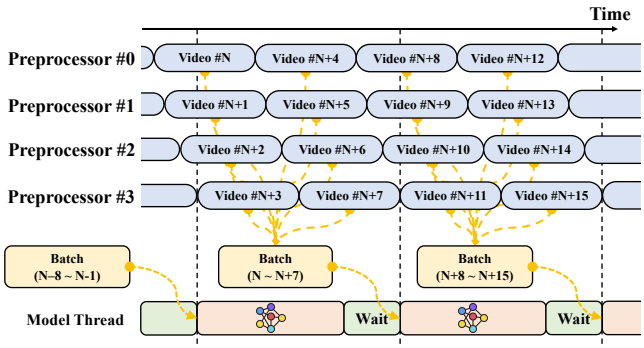


Figure 1: Time flow diagram of video preprocessing and model execution.

Framework	Utilized Hardware	Modality	Use Case
PRESTO (Ours)	CPU & GPU	Video	Inference
DALI [11]	GPU	Agnostic	Infer/Training
PyTorch DL [12]*	CPU	Agnostic	Infer/Training
FusionFlow [6]	CPU & GPU	Image	Training
SAND [2]	CPU	Video	Training
FastFlow [13]	CPU	Image/Audio	Training
Revamper [7]	CPU	Image	Training
CoorDL [10]	CPU	Image/Audio	Training

Table 1: Comparison of preprocessing frameworks for AI tasks. *DataLoader with FFmpeg/OpenCV decoding.

executor, which performs inference on GPUs. After the inference, the results are returned to the respective clients.

Achieving high throughput while maintaining *total inference latency* within the Service Level Objective (SLO) is essential for maximizing resource efficiency and reducing operational costs for inference service providers. Total inference latency (hereafter referred to simply as total latency) is the sum of (i) the preprocessing latency—mainly video decoding and frame augmentations—and (ii) the model execution latency on GPUs. Meanwhile, system *throughput*, expressed as the number of completed video requests per second (rps), is maximized when GPUs remain fully utilized without stalling. If preprocessing lags behind, the GPU must wait for the decoded frames, leading to longer total latency and idle GPU cycles as illustrated in Figure 1. Therefore, preprocessing must be fast and scalable enough to feed preprocessed batches to the model executor continuously.

Existing preprocessing frameworks. Because preprocessing can be computationally intensive, a number of frameworks have been developed to optimize data preprocessing. PyTorch DataLoader [12] is a de facto standard used in today's AI pipelines; it uses multiple CPU cores to parallelize loading and data augmentation while also pipelining the preprocessing in the CPU and the batch execution in the GPU. For video-based AI tasks, PyTorch DataLoader typically leverages CPU-based software decoders such as FFmpeg or OpenCV.

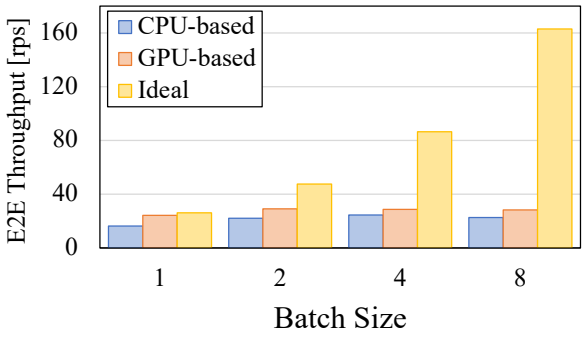


Figure 2: End-to-end throughput with CPU/GPU-based preprocessing compared to the ideal case.

Another popular solution is NVIDIA's Data Loading Library (DALI) [11], a GPU-accelerated library for deep learning workflows. DALI supports image and video decoding and augmentation (e.g., resizing, cropping, normalization). For video workloads, DALI leverages NVDEC, a dedicated hardware decoder built into NVIDIA GPUs.

Optimizing preprocessing pipeline. Several studies [2, 6, 7, 10, 13] have explored optimizing preprocessing pipelines specifically tailored for *training* workloads, as shown in Table 1. SAND [2], CoorDL [10], and Revamper [7] introduce caching to reduce redundant data preprocessing and I/O, enabling reuse of partially processed data and thus reducing GPU idle times during training. FastFlow [13] offloads preprocessing to remote CPU clusters, while FusionFlow [6] leverages both CPU and GPU resources to optimize image-based training.

Despite their contributions, these frameworks mainly address preprocessing optimizations for *training* and generally target image/audio modalities rather than *video inference*. In contrast, *video-based AI inference* poses significantly greater preprocessing challenges due to the compute-intensive nature of video decoding and real-time inference requirements. To the best of our knowledge, this paper is the first to present a preprocessing framework specifically designed for *video-based AI inference*.

3 Key Observations

In this section, we demonstrate how video preprocessing becomes a critical bottleneck in video-based AI inference, substantially limiting the inference throughput and increasing the latency. To show this, we benchmark HD-VILA [14], one of the popular video captioning models. The detailed setup is explained in §6. We focus on two widely adopted preprocessing approaches for inference, as shown in Table 1: CPU-based (PyTorch DataLoader + FFmpeg video decoder) and GPU-based (NVIDIA DALI) preprocessing frameworks.

End-to-end throughput is capped by preprocessing. For optimal GPU utilization and maximum throughput, the batch size (i.e., the number of videos being processed) should be increased until the GPU saturates. Figure 2 shows that, as batch size grows, the ideal (model-only) throughput—assuming that preprocessing latency can

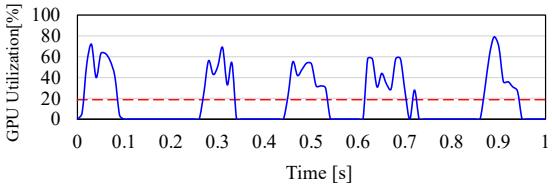
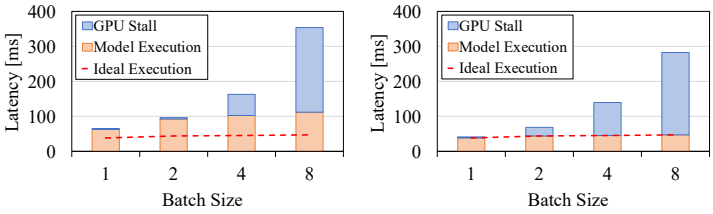


Figure 3: GPU utilization of an inference pipeline with CPU-based preprocessing. The red line indicates the average GPU utilization.



(a) CPU-based preprocessing **(b) GPU-based preprocessing**

Figure 4: Preprocessing latency by each preproc. framework.

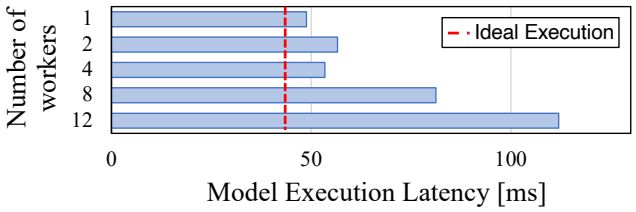


Figure 5: Model execution latency increases with more preprocessing workers due to CPU contention.

be hidden—increases almost proportionally, confirming that GPU execution scales well. However, the actual end-to-end throughput drifts ever farther from this ideal, with the gap widening steadily. Figure 3 shows that even at batch size 8, the GPU reaches only about 70% peak and 18.7% average utilization, suggesting that the real bottleneck lies in the preprocessing pipeline, not in the GPU. As a result, the two most common preprocessing approaches—CPU-based and GPU-based—achieve only 13.9% and 17.4% of the GPU-bound (model-only) throughput at batch size 8, respectively. Note that since video decoding is already operating at full capacity, increasing the batch size no longer boosts the system throughput.

Why is video preprocessing expensive? Video preprocessing in deep learning workload is far more expensive than the preprocessing for image-based tasks [2]. This is due to two key reasons: First, video decoding incurs substantial computational overhead. Unlike still images, modern video files store multiple compressed images with motion vectors and frequency-domain residuals; reconstructing each frame therefore entails entropy decoding, inverse transforms, motion compensation, and in-loop filtering for every block.

Second, we have to decode much more frames than the model actually requires. This is because most video AI models need only a small number of relevant frames (e.g., key frames) to capture the necessary context, not every intermediate or redundant frame. For instance, HD-VILA [14] actually uses approximately 20% of the decoded frames. Unfortunately, due to the sequential nature of compressed video formats, decoding these necessary frames often requires processing a large number of preceding frames as well. As a result, this compute-heavy step dominates preprocessing and stalls the model threads, as illustrated in Figure 1.

Next, we delve deeper into the specific limitations of each preprocessing framework's approach.

Limitations of CPU-only preprocessing. CPU-only preprocessing suffers from two fundamental limitations. First, software-based video decoding is computationally expensive, which significantly limits video decoding throughput. Because the throughput of CPU-based decoding does not match that of model inference, using a larger batch size of 8 results in 6.78 × longer preprocessing latency than a batch size of 1, as shown in Figure 4a.

Second, adding more preprocessing workers introduces CPU resource contention with the model execution thread, ultimately increasing the inference latency. Figure 5 demonstrates that such contention raises model execution time by up to 2.4× above the ideal baseline. These limitations make CPU-based pipelines fundamentally unsuitable for high-throughput video-based AI inference.

Limitation of GPU-only preprocessing. Although dedicated hardware such as NVDEC decodes faster than CPU-based decoders, its throughput still falls short of the model's demand. For instance, NVDEC can decode 4.73 × faster (24.29 rps) than the CPU-based software decoder (5.13 rps) for a single 720p resolution video. However, NVDEC is still far from the optimal model throughput. To measure its limits, we implement our optimized multi-threaded NVDEC pipeline (§5.2) and confirm that NVDEC can achieve up to 62.5 videos per second. For seamless model execution, we still need to speed this up by 2.59 × higher.

Moreover, we observe that the state-of-the-art GPU-based preprocessing framework performs surprisingly worse than our optimized multi-threaded NVDEC pipeline, despite supporting NVDEC. As shown in Figure 4b, with a batch size of 8, DALI experiences 6.88 × higher processing latency compared to the case with a batch size of 1. This is because DALI supports only a single-threaded control to utilize NVDEC, which can process one video decoding at a time. Even more limiting, this single host thread is also responsible for post-decoding tasks such as YUV-to-RGB conversion and copying results into tensor memory. As a result, when using DALI, the NVDEC engine often sits idle, leading to much lower throughput than the model can consume.

Our findings show that relying solely on either the CPU or GPU for video preprocessing falls short of meeting high-throughput demands in video-based AI inference. A more efficient preprocessing strategy and a supporting framework are required to close this gap.

4 Approach: Hybrid CPU-GPU processing

A promising way to overcome the limitations of CPU-only or GPU-only preprocessing is to adopt a hybrid CPU-GPU approach, where

both CPU and GPU share video decoding tasks. This allows us to take advantage of the best of both worlds: utilizing fast GPU decoding hardware while also leveraging the CPU, which offers greater flexibility and can be scaled more easily by utilizing idle cores often available in deep learning workloads.

Goal. The primary objective of PRESTO is to eliminate bottlenecks in the preprocessing pipeline so that the GPU can remain busy for model execution. By collaboratively using both NVDEC (GPU-based) and FFmpeg (CPU-based) decoders, this paper aims to effectively remove preprocessing bottlenecks in video-based AI model inference and maximize overall pipeline efficiency.

Design considerations. While hybrid CPU-GPU preprocessing is promising, implementing this strategy and achieving meaningful throughput over existing methods requires addressing three design considerations:

- **Efficient resource allocation and scheduling**: To fully harness the synergy of CPU-GPU collaboration, a scheduler must allocate resources and orchestrate preprocessing tasks between the CPU and GPU based on system load and video characteristics. This scheduling logic is essential to the framework's flexibility and efficiency. Also, we need to develop knobs to control decoder-level resource allocation for our fine-grained scheduling.
- **Maximizing decoding performance**: Simply using the video decoding approach from existing data processing frameworks is insufficient to fully realize high throughput with the hybrid approach. We should develop a new decoding strategy tailored for video-based AI inference to maximize the decoding performance on both the CPU and GPU sides.
- **GPU memory management**: Modern video models already occupy tens of gigabytes of device memory, and their footprints continue to grow with every architectural advance. When using GPU resources for preprocessing, multiple video decoders share GPU memory with the video AI model, leaving less memory available for inference. Without careful memory management, this sharing creates significant memory pressure on limited GPU resources. Developing efficient memory management strategies is essential.

5 PRESTO Design

Figure 6 sketches end-to-end pipeline of PRESTO. Black arrows denote the video-data flow, whereas green arrows denote the GPU-buffer flow reused by the GPU memory pool. By orchestrating CPU and GPU resources along these two orthogonal flows, PRESTO removes the preprocessing bottleneck that harms video-based AI inference while achieving high inference throughput.

Video-data flow. Upon receiving an inference request, *Hybrid Preprocessing Scheduler* selects the best preprocessor available. The designated preprocessor decodes the video with *Pipelined Selective Decoding* and packs processed frames into a batch. Finally, the model executor feeds the generated batch into a model to obtain the final results.

GPU-buffer flow. *GPU Memory Manager* handles GPU memory buffer allocation to enable efficient reuse and avoid redundant allocations. It provides GPU memory buffers to store preprocessed

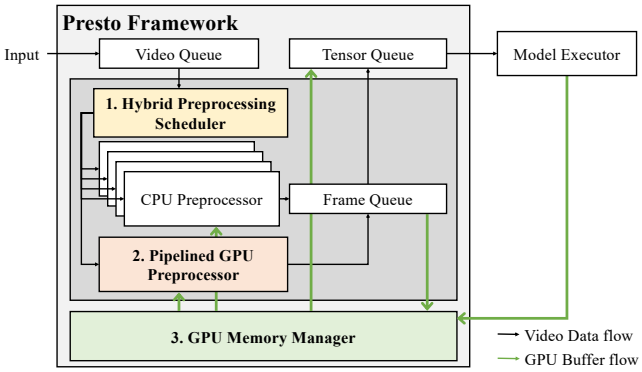


Figure 6: PRESTO Design Overview.

frames generated by CPU or GPU-based preprocessors. Once the frames are processed, the manager allocates a contiguous buffer to store the packed batch, which is then enqueued into the tensor queue. After the model executor finishes processing the batch, the associated frame buffers are immediately returned to the manager for reuse.

5.1 Hybrid Preprocessing Scheduler

To efficiently allocate preprocessing tasks across heterogeneous CPU and GPU resources, we employ a simple yet effective policy that minimizes the data waiting time while maximizing the hardware utilization. We consider both the current workload and the processing capability of each resource by utilizing a normalized queue length metric that accounts for both the pending work and the historical processing throughput of each preprocessor. The metric is defined as follows:

$$\tilde{Q}_i = \frac{Q_i}{\tau_i}, \text{ where, } \tau_i = \frac{N_i}{T_i}, \quad T_i = \sum_{j=1}^{N_i} t_{i,j}$$

Here, i indexes one of the preprocessors in PRESTO, Q_i is the number of pending videos, and τ_i is the average throughput, computed as $\tau_i = N_i T_i$, where N_i is the number of videos processed so far and $T_i = \sum_{j=1}^{N_i} t_{i,j}$ is the total processing time. $t_{i,j}$ denotes the latency to preprocess the j-th video assigned to preprocessor i. Our scheduler selects the one with the shortest normalized queue $\tilde{Q}_i$.

In future work, we aim to investigate scheduling policies that more directly optimize for overall system throughput.

5.2 Pipelined Selective Decoding

PRESTO maximizes the decoding performance of each individual preprocessor through selective frame decoding tailored for video-based AI models and a highly efficient pipelined GPU decoder.

Selective decoding. A key insight in PRESTO decoding is that not all frames need to be decoded. As discussed in §3, video-based AI models typically use only 10–20% of the frames in a video—often sampling one frame every k frames. Based on this observation, PRESTO employs selective decoding; it jumps to the closest key frame that precedes the target frame required for inference, skipping all unnecessary intermediate frames. In detail, upon receiving a

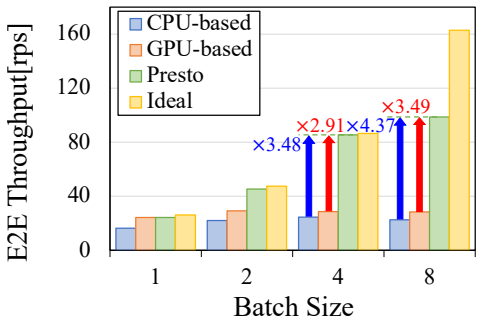


Figure 7: End-to-end inference throughput under the optimal configuration for each batch size.

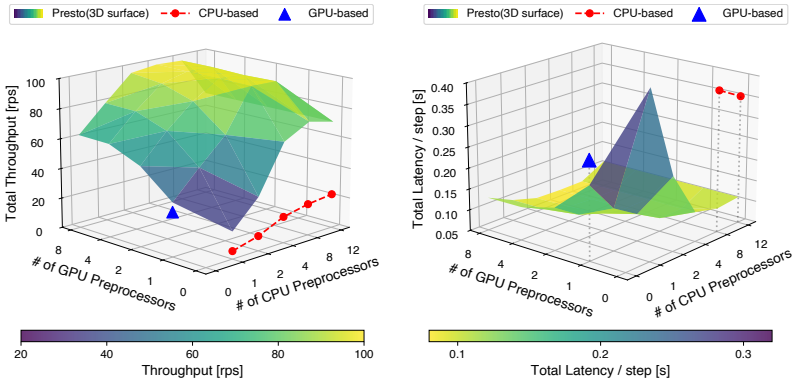


Figure 8: Performance with different configurations.

request, the preprocessor seeks the key frame closest to the target frame's Packet Time Stamp (PTS), which indicates the temporal position of the frame in the video stream, bypassing all earlier packets. Since finding a non-key frame's exact PTS without decoding is impossible, we estimate the target frame's PTS using its frame index and video metadata (i.e., frame rate and timebase).

Pipelined video decoding (GPU). Another design insight in PRESTO is that the GPU-based decoding workflow can be decomposed into distinct stages (pre-decode, decode, and post-decode), many of which are amenable to pipelining. By exploiting this structure, PRESTO decomposes this decoding pipeline and enables multiple preprocessor threads to share the limited NVDEC engine.

Specifically, PRESTO decomposes the pipeline into bitstream I/O and demuxing, packet submission, decoding, YUV-to-RGB conversion, tensor formatting, and uploading the tensors to GPU memory. Among these, only the decoding process needs to run on the NVDEC engine, while the others run on CUDA cores and CPUs. Leveraging this separation, PRESTO enables the concurrent processing of multiple videos. While one video occupies the NVDEC engine for decoding, other videos can simultaneously progress through bit-stream reads or post-processing kernels on CUDA cores and CPUs.

5.3 GPU Memory Manager

Inference with video-based AI models alone already consumes tens of gigabytes of device memory for storing model weights and temporary activations, with their footprints growing further with each architectural advancement. Under such tight memory constraints, repeatedly invoking `cudaMalloc` and `cudaFree` for each decoded frame or tensor fragment leads to GPU memory fragmentation and non-negligible allocation latency—often taking several milliseconds per call.

PRESTO addresses both issues by pre-allocating large memory blocks and reusing them across decoding and batching stages via a dedicated GPU Memory Manager. Since memory requirements are consistent across frames, this workload-aware reuse strategy effectively reduces memory fragmentation and overall GPU memory usage, while ensuring that the reserved memory for the model remains unaffected.

Pre-allocation and reuse. GPU Memory Manager maintains an internal pool of GPU-memory buffers. When the preprocessor requests a buffer of a given size, the manager searches its pool for a block that matches the request and returns it. If no such block is available, the manager allocates a new GPU-memory region of the required size and passes it back to the preprocessor. As soon as the tensor is no longer necessary, the associated buffer is automatically returned to the Buffer Pool. By combining automatic buffer return with size-aware reuse, PRESTO prevents GPU memory leaks and virtually eliminates allocation/free overhead.

6 Preliminary Evaluation

We evaluate PRESTO using HD-VILA [14], a widely adopted video-captioning model, using the 720p version of the HD-VILA-100M dataset from the same paper. Note PRESTO is designed to support general video-based AI models, and we leave evaluation with other models as our future work. All experiments are conducted on a representative inference-server setting with 12 CPU cores and a NVIDIA RTX 4090 GPU, which emulates a Google Cloud Platform a2-ultragpu-1g instance (1× A100-80GB GPU and 12 vCPUs).

Baseline & metric. We evaluate PRESTO against two widely-used preprocessing frameworks in AI pipelines: PyTorch DataLoader with FFmpeg decoder and NVIDIA DALI with NVDEC decoder. We report the end-to-end throughput as requests per second (rps), assuming one video per request, and the total latency defined in §2.

Implementation. PRESTO is implemented in C++ and CUDA, comprising approximately 3K lines of code. The implementation runs in a Docker container and utilizes NVIDIA Video Codec SDK v12.1 and FFmpeg v6.1.

End-to-end result. Figure 7 shows that PRESTO boosts end-to-end throughput by up to 4.37× over the CPU-based pipeline and 3.49× over the GPU-based pipeline. At the same time, PRESTO reduces total latency to 130 ms; 2.72× faster than CPU-based pipeline and 2.17× faster than GPU-based pipeline. Even at batch 8, where preprocessing becomes its limiting factor, PRESTO still maintains at least a 3× advantage over both baselines across all batch sizes.

Component-wise benefit. PRESTO achieves performance gains through three components: the hybrid preprocessing scheduler, selective decoding, and pipelined decoding. In the batch 8 configuration

(Figure 7), our hybrid decoding scheduler achieves an end-to-end throughput of 8.5 rps, resulting in a 1.1× improvement over the default round-robin scheduler. For each preprocessing worker, selective decoding significantly boosts preprocessing throughput, achieving a 3.4× increase on the CPU (5.2 rps→18.0 rps) and a 1.25× increase on the GPU (28.4 rps→35.6 rps). Finally, our pipelined, multithreaded decoding increases GPU preprocessing throughput from 28.3 rps (which only supports a single worker as in DALI) to 62.5 rps (2.2× improvement) with eight workers.

Performance with different configurations. Performance of PRESTO is largely determined by the number of CPU and GPU preprocessors. To analyze the impact, we profile the end-to-end throughput and latency under varying configurations, as visualized in Figure 8. Note that latency values for CPU-based processing above 0.4s are omitted for presentation clarity. Both the throughput and latency curves exhibit a vortex-like surface, indicating that simply adding more CPU or GPU preprocessors does not ensure the best performance, and there exists an optimal balance between them. When both the numbers of CPU and GPU preprocessors are low, long decoding latency dominates the end-to-end delay, and PRESTO fails to achieve high throughput. Increasing the number of preprocessors initially improves throughput by enabling greater parallelism. However, beyond a certain point, excessive parallelism causes contention for shared resources (e.g., NVDEC engines and CPU cores), leading to increased latency and, in some extreme cases, even a drop in throughput.

Identifying the optimal configuration is an open challenge and is left for future work. Nevertheless, across a wide range of configurations, PRESTO consistently outperforms CPU- and GPU-based preprocessing in both throughput and latency.

Expected cost and carbon savings. For a workload of 1,000 video requests per second, the CPU-based pipeline requires 45 a2-ultragpu-1g instances and the GPU-based pipeline needs 35, whereas PRESTO meets the same load with only 11 instances thanks to the higher throughput processing. These 69% and 75% reduction yield an aggregate saving of roughly $165/h on Google Cloud Platform. It also lowers power consumption and consequently carbon emissions by approximately 25%, from 31.5 J to 23.4 J, compared to the CPU-based framework.

7 Conclusion & Future Directions

We present PRESTO, a hybrid CPU-GPU preprocessing framework that accelerates video-based AI inference by addressing preprocessing bottlenecks. PRESTO integrates hybrid decoding, selective frame processing, and pipelined GPU preprocessing with memory reuse. Evaluated on HD-VILA, PRESTO achieves up to 4.37 × higher throughput and 2.72 × lower latency than common baselines, enabling cost- and energy-efficient deployment of real-time video AI systems.

Beyond single-node deployments, the principles behind PRESTO can be applied to distributed inference systems operating over heterogeneous GPU clusters. In such settings, GPUs may vary in decoding capacity, compute performance, and available memory, while video requests differ in resolution, duration, task type, and SLO requirements. A natural extension of PRESTO is a network-aware runtime scheduler that co-optimizes batching and preprocessing by taking these heterogeneities into account. When preprocessing outpaces local compute, intermediate frames can be routed to higher-capacity nodes, enabling efficient load balancing while preserving per-stream SLOs. This direction opens up opportunities for scaling PRESTO to support high-throughput, real-time video AI across diverse cloud and edge environments.

Acknowledgement

We appreciate anonymous reviewers for providing constructive feedback and suggestions. We also thank Hwijoon Lim for participating in early discussions. This work is supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00340099) and Samsung Electronics Co., Ltd (No. IO221107-03428-01 and Samsung Hyper). Jaehong Kim is the corresponding author.

References

[1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. 2022. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems* 35 (2022), 23716–23736.

[2] Uitaek Hong, Hwijoon Lim, Hyunho Yeo, Jinwoo Park, and Dongsu Han. 2023. SAND: A Storage Abstraction for Video-based Deep Learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems*. 16–23.

[3] Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, et al. 2023. Planning-oriented autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 17853–17862.

[4] Bin Huang, Xin Wang, Hong Chen, Zihan Song, and Wenwu Zhu. 2024. Vtimellm: Empower llm to grasp video moments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14271–14280.

[5] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. 2024. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246* (2024).

[6] Taeyoon Kim, ChanHo Park, Mansur Mukimbekov, Heelim Hong, Minseok Kim, Ze Jin, Changdae Kim, Ji-Yong Shin, and Myeongjae Jeon. 2023. Fusionflow: Accelerating data preprocessing for machine learning with cpu-gpu cooperation. *Proceedings of the VLDB Endowment* 17, 4 (2023), 863–876.

[7] Gyewon Lee, Irene Lee, Hyeonmin Ha, Kyunggeun Lee, Hwarim Hyun, Ahnjae Shin, and Byung-Gon Chun. 2021. Refurbish your training data: Reusing partially augmented samples for faster deep neural network training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 537–550.

[8] Ze Liu, Jia Ning, Yue Cao, Yixuan Wei, Zheng Zhang, Stephen Lin, and Han Hu. 2022. Video swin transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 3202–3211.

[9] Tengchao Lv, Yupan Huang, Jingye Chen, Yuzhong Zhao, Yilin Jia, Lei Cui, Shuming Ma, Yaoyao Chang, Shaohan Huang, Wenhui Wang, et al. 2023. Kosmos-2.5: A multimodal literate model. *arXiv preprint arXiv:2309.11419* (2023).

[10] Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. 2021. Analyzing and mitigating data stalls in DNN training. *Proc. VLDB Endow.* 14, 5 (Jan. 2021), 771–784. https://doi.org/10.14778/3446095.3446100

[11] NVIDIA. 2023. NVIDIA DALI. https://developer.nvidia.com/blog/why-automatic-augmentation-matters

[12] PyTorch. 2021. PyTorch DataLoader. https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

[13] Taegeon Um, Byungsoo Oh, Byeongchan Seo, Minhyeok Kweun, Goeun Kim, and Woo-Yeon Lee. 2023. Fastflow: Accelerating deep learning model training with smart offloading of input data pipeline. *Proceedings of the VLDB Endowment* 16, 5 (2023), 1086–1099.

[14] Hongwei Xue, Tiankai Hang, Yanhong Zeng, Yuchong Sun, Bei Liu, Huan Yang, Jianlong Fu, and Baining Guo. 2022. Advancing high-resolution video-language representation with large-scale video transcriptions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5036–5045.

[15] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Fucheng Weng, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. 2022. Bytetrack: Multi-object tracking by associating every detection box. In *European conference on computer vision*. Springer, 1–21.