FluxShard: Distributed Motion-Aware Cache Remapping for Real-Time Video Analytics

IEEE Publication Technology Department

***Abstract*—Edge-cloud video analytics caches intermediate features across consecutive frames to cut redundant computation and transmission. Existing methods operate at whole-scene granularity—reusing or invalidating entire feature maps—and assume a roughly stationary scene. Even moderate motion misaligns the cache with the current frame, triggering widespread invalidation although most content has merely shifted rather than truly changed. The root cause is a *granularity mismatch*: the cache is treated as an indivisible unit, yet real-world motion is local and heterogeneous.**

We present FluxShard, a motion-aware edge-cloud analytics system that elevates codec-level motion vectors (MVs) into a first-class control signal for feature caching. FluxShard decomposes the cached feature map into block-level shards that flow with observed motion, realigning reusable content and isolating genuinely changed regions as a minimal recomputation set. A *Receptive-Field Alignment Principle* guarantees bit-equivalent correctness of MV-guided reuse across convolutional layers, and a *profiling-driven truncation policy* sustains high sparsity under growing motion with negligible accuracy loss. At runtime, FluxShard adaptively routes the sparse residual workload between edge and cloud based on network conditions and device load. Evaluation on real-world dynamic video sequences spanning object detection and instance segmentation shows that FluxShard achieves up to 92% bandwidth reduction, 5.5× speedup, and ∼99% accuracy retention over state-of-the-art baselines.

***Index Terms*—Video analytics, robotics, edge-cloud collaborative system, CNN inference acceleration**

I. Introduction

Video analytics underpins latency-critical applications such as embodied intelligence [22], [7], aerial drones [1], [16], augmented reality [23], and smart surveillance [10]. Processing solely on edge devices avoids transmission delay but is constrained by limited compute and energy; offloading to the cloud provides ample resources but introduces network latency. A natural middle ground is to *reuse* previously computed features across consecutive frames: since adjacent frames share substantial visual content, caching and reusing unchanged results reduces both redundant computation and transmission.

Existing reuse methods, however, operate at the granularity of the *whole scene* and are therefore brittle under motion. Pipeline-level approaches [13], [5] cache entire intermediate feature maps and reuse them only when successive frames are globally similar. Delta-based methods [2], [15], [19] maintain a scene-level reference and update it via pixel-wise differences; MotionDeltaCNN [18] extends this with a single global homography. Because all these approaches treat the cache as an indivisible entity, any camera ego-motion or object movement triggers widespread cache invalidation—even when most content has merely *shifted* rather than changed. The root cause is a *granularity mismatch*: the cache is monolithic, yet real-world motion is local and heterogeneous.

Block-level motion vectors (MVs) from standard video codecs offer a natural remedy. Extracted as a free byproduct of decoding, MVs capture per-region displacement between consecutive frames and can therefore distinguish spatial shift from genuine content change. Re-indexing cached features according to these displacements recovers reusable content that whole-scene methods would discard, while regions where MVs are inherently insufficient—non-rigid deformation, disocclusion, newly appearing content—are isolated as a minimal *residual set* that truly requires recomputation.

Realizing this idea, however, introduces two challenges. (1) the **receptive field inconsistency** problem: even when cached features are correctly aligned via MVs, directly reusing the aligned output of a layer that aggregates over a spatial neighborhood of input (i.e., receptive field) is not always valid, because heterogeneous motion can assemble a neighborhood that differs from the one used to produce the cached output, silently corrupting results. Naïvely detecting such mismatches requires checking the full receptive field at every output position of every layer, incurring prohibitive cost rivaling recomputation itself. (2) the **sparsity decay** problem:the residual set tends to expand through cascaded layers as each layer's receptive field causes recomputed positions to bleed into their neighbors, progressively eroding the efficiency advantage of selective recomputation as motion intensity grows.

We propose **FluxShard**, a motion-aware edge-cloud video analytics system that treats the feature cache as block-level *shards* flowing freely with motion vectors. FluxShard uses MVs as a first-class control signal to align cached features with the current frame, isolate the minimal residual set, and dynamically route work—local sparse update on the edge or selective offloading of only residual regions to the cloud—based on residual size, network conditions, and device load.

To address receptive field inconsistency, FluxShard introduces the **Receptive Field Alignment Principle (RFAP)** (§**??**), which folds the per-layer, per-position neighborhood check into a single lightweight pass over the input-level MV field. RFAP produces a conservative reuse mask at each layer in time linear in the input resolution, guaranteeing bit-equivalent correctness without per-layer verification overhead.

To counter sparsity decay, FluxShard employs a **profiling-driven truncation policy** (§**??**). Offline, we characterize the tightest accuracy-preserving truncation thresholds across a range of motion intensities; at runtime, the system indexes into this mapping based on observed motion, dynamically

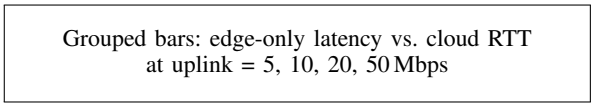

Figure 1. Per-frame latency under varying uplink bandwidth. Neither pure-edge nor pure-cloud meets real-time across all conditions.

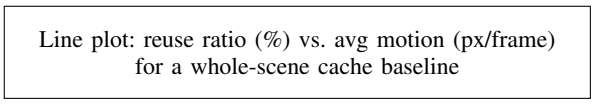

Figure 2. Cache reuse ratio versus per-frame motion. Whole-scene granularity causes reuse to collapse under moderate motion.

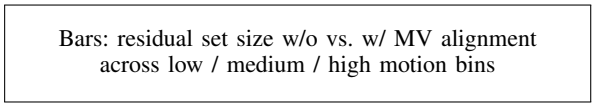

Figure 3. MV alignment eliminates displacement-induced false misses, cutting the residual set by **XX–XX**%.

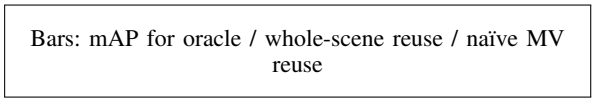

Figure 4. Naïve MV-aligned reuse destroys accuracy due to receptive field inconsistency, even when the residual set is small.

modulating truncation to sustain high sparsity with negligible accuracy loss.

In summary, this paper makes the following contributions:

- We identify granularity mismatch as the key limitation of whole-scene cache reuse and show that codec-level motion vectors provide the right abstraction to overcome it.
- We propose FluxShard, a system that decomposes the feature cache into motion-aligned shards and co-optimizes sparse recomputation with adaptive edge-cloud workload routing.
- We establish the Receptive Field Alignment Principle for bit-equivalent MV-guided feature reuse across layers, and a profiling-driven truncation policy that sustains high sparsity under varying motion with bounded accuracy loss.
- We evaluate FluxShard on object detection and instance segmentation with YOLOv11 [12] on two dynamic video benchmarks (DAVIS [20], 3DPW [21]). Compared to the strongest baseline, FluxShard achieves up to $5.5\times$ end-to-end speedup and 92% bandwidth reduction while retaining over 99% of full-model accuracy.

II. Challenges and Motivation

A. The Edge-Cloud Dilemma

Edge-cloud video analytics must navigate two competing bottlenecks. Running YOLOv11m [12] on an NVIDIA Jetson Xavier NX takes **XX** ms per 1080p frame, far exceeding a 33 ms real-time budget. Offloading every frame removes the compute bottleneck but introduces a transmission one: JPEG-compressed 1080p at 30 fps sustains **XX** Mbps, routinely exceeding edge uplink capacity. Figure 1 quantifies this tension.

Feature cache reuse bridges this gap: when consecutive frames share content, cached results substitute for fresh computation. However, existing mechanisms [13], [5], [19], [18] treat the cache as an indivisible whole-scene entity—whether the decision is binary (reuse or recompute the entire frame) or pixel-level (propagate a dense difference map). This granularity breaks down under motion. Figure 2 shows that even modest displacement triggers widespread invalidation: the reuse ratio drops from **XX**% on near-static sequences to below **XX**% once motion exceeds **XX** px/frame, despite most content being merely shifted rather than truly changed.

Takeaway. Cache reuse is essential, but whole-scene granularity cannot tolerate motion. A finer-grained, motion-aware mechanism is needed.

B. Motion Vectors: Opportunity and Correctness Failure

H.264/H.265 codecs estimate a per-block displacement (d_x, d_y) as part of normal encoding, providing per-region motion information at *zero additional cost*. By shifting cached feature blocks according to these motion vectors (MVs), displaced content is realigned before differencing, and only genuinely changed regions—dis-occlusion, deformation, new objects—remain in the *residual set*. Figure 3 confirms MV alignment reduces the residual set by **XX–XX**% across motion intensities: most whole-scene cache misses stem from displacement, not true change.

Despite the smaller residual set, directly reusing MV-aligned features produces catastrophic accuracy loss. Figure 4 shows mAP drops from **XX**% to **XX**% under naïve MV reuse. The root cause is *receptive field inconsistency*: layers with receptive field size > 1 aggregate spatial neighborhoods. When adjacent blocks carry different MVs, re-indexing assembles patches that were never contiguous in the original frame; the cached output was computed from a different neighborhood. This resulting error compounds through cascaded layers.

Using the codec's own residual is not an alternative: H.264/H.265 residual coding targets perceptual quality, not inference accuracy. At lower bitrates, quantization artifacts propagate through DNN layers and degrade accuracy [] TODO: add reference.

Takeaway. MV alignment eliminates most false cache misses, but exploiting it requires a correctness guarantee under heterogeneous per-block motion. This is the *first challenge* our design must address.

C. Sparsity Decays Through Cascaded Layers

Even with correct reuse, a second challenge erodes efficiency. Each convolutional layer with kernel radius r reads r positions beyond every active position, expanding the residual set by a margin of r per side per layer. Over L layers the expansion compounds: a single active input position inflates into a region of radius $\sum_l r_l$ at the final layer.

Figure 5 traces per-layer residual ratio through YOLOv11m. A modest input residual ratio of **XX**% grows past **XX**% by mid-network and saturates near 100% before the last layer.

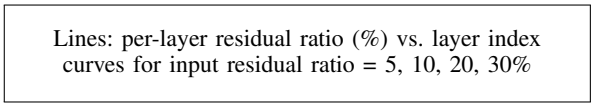

Figure 5. Receptive field bleed causes the residual set to expand at each layer, approaching full recomputation before the final layer.

At that point selective recomputation offers no advantage over full inference.

Takeaway. Without an explicit mechanism to arrest receptive field bleed, selective recomputation degenerates into full recomputation under real-world motion. A sparsity-preserving strategy with bounded accuracy loss is the *second challenge* our design must address.

III. System Overview

This section presents the FluxShard system. We first define the system model and optimization objective (§III-A), then describe the end-to-end pipeline that addresses the two challenges identified above (§III-B).

A. System Model

We consider an edge-cloud video analytics system in which an edge device (e.g., an NVIDIA Jetson) receives a live video stream from a co-located camera and a remote cloud server provides supplementary compute capacity. The two endpoints communicate over a bandwidth-limited uplink with round-trip latency that varies over time. Each endpoint hosts an identical copy of the inference model and maintains a per-layer feature cache that stores the output produced during its most recent inference. For each incoming frame, the system makes a *dispatch decision*: whether to run inference locally at the edge or offload it to the cloud.

a) Notation.: Let $\{I_t\}$ denote the incoming frame sequence. We write $\mathbf{m}_t$ for the block-level motion vector (MV) field extracted from the codec when decoding frame I_t; each entry maps a block position in I_t to its corresponding position in the reference frame.

The DNN $\mathcal{N}$ comprises L layers indexed by $l \in \{1, \ldots, L\}$. Layer l takes an input feature map $\mathbf{F}^l \in \mathbb{R}^{H_l' \times W_l' \times C_l'}$ and produces an output feature map $\mathbf{O}^l \in \mathbb{R}^{H_l \times W_l \times C_l}$; by convention $\mathbf{F}^1 = I_t$. Since our analysis concerns only spatial positions, we omit the channel dimension hereafter. The output of a layer at spatial position (i, j) depends on a set of input positions called its *receptive field*, denoted $\mathcal{R}^l(i, j) \subseteq \{1, \ldots, H_l'\} \times \{1, \ldots, W_l'\}$, with radius r_l. For convolutional and linear layers, $\mathbf{w}^l$ denotes the weight tensor applied over this receptive field.

Each endpoint caches the output of every layer from its most recent inference; we write $\hat{\mathbf{O}}^l$ for the cached output at layer l and $\hat{\mathbf{F}}^l$ for the corresponding cached input. Given the MV field, position (i, j) in the current frame maps to a cached position $(\hat{\imath}, \hat{\jmath})$; likewise, each input position $(p, q) \in \mathcal{R}^l(i, j)$ maps to its cached counterpart $(\hat{p}, \hat{q})$.

A *shard* is a block-sized unit of a feature map whose spatial extent corresponds to one entry in the MV field. At deeper layers with reduced spatial resolution, neighboring MVs that map to the same shard are aggregated and their magnitudes divided by the stride factor, so that each shard retains a one-to-one correspondence with a single displacement measured in feature-map coordinates.

b) Assumptions.: The inference result (e.g., detections, segmentation masks) is consumed at the executing endpoint or forwarded to a co-located downstream service. Downstream processing is typically lightweight (e.g., issuing an alert, logging an action label), and the processed outcome is negligible in size; we therefore treat result delivery as outside the latency-critical path and exclude it from our optimization scope.

c) Objective.: Let $d_t \in \{\text{edge}, \text{cloud}\}$ denote the dispatch decision for frame I_t. The per-frame end-to-end latency $T(d_t)$ comprises local computation, data transmission, and remote computation as applicable. Let $\mathcal{A}(d_t)$ denote the task accuracy achieved under decision d_t and $\mathcal{A}^*$ the accuracy of full recomputation without caching. In general, latency reduction techniques trade off accuracy for speed, so $\mathcal{A}(d_t)$ depends on the specific optimizations applied at the selected endpoint. Our objective is:

$$\min_{d_t} \; T(d_t) \quad \text{s.t.} \quad \mathcal{A}(d_t) \;\geq\; \alpha\, \mathcal{A}^*. \tag{1}$$

B. Pipeline

Figure 6 illustrates the end-to-end FluxShard pipeline. Each endpoint maintains a *dispatch layer*: a lightweight bookkeeping structure that stores (i) an *input cache*—the input used in the endpoint's most recent inference—and (ii) an *accumulated MV field* from that cached input to the present frame. The dispatch layer enables per-endpoint reusability estimation without running the model.

When an encoded frame arrives at the edge, FluxShard proceeds in four stages.

a) Stage 1: MV extraction.: Block-level motion vectors $\mathbf{m}_t$ are extracted from the video codec as a free byproduct of decoding, capturing per-region motion at no additional cost. Both dispatch layers incorporate $\mathbf{m}_t$ into their respective accumulated MV fields.

b) Stage 2: Reusability estimation.: Each dispatch layer uses its accumulated MV field to shift its cached input, aligning displaced content with the current frame. A per-block comparison between the aligned cache and the decoded frame identifies positions with non-negligible difference; these positions form the *residual set*. From the residual set size, the edge dispatch layer estimates local sparse inference latency, while the cloud dispatch layer estimates the cost of transmitting the MV field and residual pixels plus remote inference under current network conditions.

c) Stage 3: Dispatch decision.: The frame is assigned to the endpoint with the lower estimated latency, implicitly adapting to motion complexity, network bandwidth, and endpoint load.

d) Stage 4: Inference and cache update.: The selected endpoint applies its accumulated MV field to rearrange cached shards across all layers and fills in residual pixels at the input, producing the updated input; its input cache is then replaced and its accumulated MV field reset. The residual set is propagated layer by layer through the model: at each layer, the

FluxShard end-to-end pipeline.

Left: edge device (camera → codec → MV extraction → dispatch layer).
Center: dispatch decision (edge local vs. cloud offload).
Right: selected endpoint (shard alignment → RFAP → truncation → sparse inference → cache update).

Arrows show MV field and residual pixels transmitted to cloud when offloading.

Figure 6. Overview of the FluxShard pipeline. Numbered stages correspond to the description in §III-B.

Receptive Field Alignment Principle (§**??**) identifies positions that can be safely reused despite heterogeneous motion, and a truncation policy (§**??**) suppresses positions whose residual magnitude is sufficiently small. Only the remaining positions are recomputed; their outputs are merged into the layer's feature cache. The final result is produced by fusing reused and freshly computed features at the last layer.

The non-selected endpoint retains its accumulated MV field, allowing its cache to span a longer interval for future reuse. When the server is selected, the edge transmits only the accumulated MV field and the residual pixels; a lightweight *receive layer* on the server mirrors the dispatch layer logic to reconstruct the full input before inference.

IV. System Design

The preceding sections establish that per-block MVs can, in principle, re-index cached features to recover shifted content. As a preliminary for our design, this section formalizes the layer-level reuse criterion and identifies the obstacle that per-block heterogeneous MVs pose to correct feature cache reuse.

Layer-local reuse criterion. Consider a layer l that computes an output feature map $\mathbf{O}^l \in \mathbb{R}^{H_l \times W_l \times C_l}$ from an input feature map $\mathbf{F}^l \in \mathbb{R}^{H'_l \times W'_l \times C'_l}$. Since our analysis concerns only spatial positions, we omit the channel dimension hereafter. The output at position (i, j) depends on a set of input positions called its *receptive field* $\mathcal{R}^l(i, j) \subseteq \mathbf{F}^l$. The feature cache stores the previous frame's output $\hat{\mathbf{O}}^l$; a motion vector maps (i, j) to a cached position $(\hat{\imath}, \hat{\jmath})$ whose value was produced from a corresponding receptive field in the previous input $\hat{\mathbf{F}}^l$. Reuse at (i, j) is valid when the output discrepancy stays within a task-driven tolerance:

$$\left|\mathbf{O}^l(i,j) - \hat{\mathbf{O}}^l(\hat{\imath},\hat{\jmath})\right| \;\le\; \tau. \tag{2}$$

This condition is universal across layer types.

For linear layers (convolution, addition, etc.), let $\mathbf{w}^l$ collect the coefficients applied over $\mathcal{R}^l(i, j)$; by linearity the output error satisfies

$$\left|\mathbf{O}^l(i,j) - \hat{\mathbf{O}}^l(\hat{\imath},\hat{\jmath})\right| \;\le\; \max_{(p,q)\,\in\,\mathcal{R}^l(i,j)} \|\mathbf{w}^l\|_1 \cdot \left|\mathbf{F}^l(p,q) - \hat{\mathbf{F}}^l(\hat{p},\hat{q})\right|, \tag{3}$$

where $(\hat{p}, \hat{q})$ is the MV-mapped counterpart of (p, q). Thus (2) is guaranteed whenever the input patch satisfies

$$\max_{(p,q)\,\in\,\mathcal{R}^l(i,j)} \left|\mathbf{F}^l(p,q) - \hat{\mathbf{F}}^l(\hat{p},\hat{q})\right| \;\le\; \frac{\tau}{\|\mathbf{w}^l\|_1}. \tag{4}$$

For nonlinear activations such as ReLU or sigmoid that are 1-Lipschitz, Eq. (2) at the input side directly implies the same bound at the output, so no additional margin is needed.

In practice, the check in Eq. (4) is the one executed at runtime for each linear or nonlinear activation layer, while Eq. (2) serves as the invariant maintained across the entire layer stack. Crucially, every evaluation uses only layer l's own input and cached output; it never recurses back to the model input.

When can reuse decisions be shared across layers? Amortizing the reuse decision across layers eliminates the per-layer input check in Eq. (4), which can itself be costly. When $|\mathcal{R}^l| = 1$ (e.g., 1×1 convolution, pointwise activation), such sharing is automatic: reused positions carry zero difference into the next layer and satisfy any threshold trivially, while recomputed positions carry nonzero difference and cannot pass the tighter threshold $\tau/\|\mathbf{w}^{l+1}\|_1 \le \tau$. The reuse map is therefore inherited by all subsequent $|\mathcal{R}|{=}1$ layers at the same resolution with same MVs without re-evaluation.

Once $|\mathcal{R}^l| > 1$ (e.g., 3×3 convolution), sharing breaks down. Each output now aggregates a spatial neighborhood whose elements may have been relocated from arbitrary, disjoint positions by heterogeneous motion vectors. Even if every individual input is reused with zero difference, the spatial composition within the receptive field differs from the one that produced the cached reference output; the difference in Eq. (4) therefore no longer measures true content change, invalidating the reuse criterion. Moreover, any layer that alters spatial resolution (e.g., downsampling) changes the MV-to-position mapping, invalidating reuse decisions from earlier layers. A straightforward approach to maintaining correctness therefore requires an independent check at every layer with $|\mathcal{R}^l| > 1$ and every output position, a cost that can rival the computation it seeks to avoid.

Delta methods as a degenerate but tractable special case. Delta methods [19], [18] sidestep this limitation by restricting every block to a single shared MV: zero for a static camera, or a global constant after homography alignment. Under this uniformity, the MV-mapped correspondence between the current and cached receptive field patches reduces to the same fixed displacement at every output position. If every position within a layer-l receptive field was itself reusable at layer $l{-}1$, the two patches in Eq. (4) are identical and the output position is reusable as well. The reuse map therefore only needs to be seeded at the input, where it is simply the set of unchanged pixels, and propagated through subsequent layers via receptive field inclusion.

This simplicity comes at a cost: a single global MV cannot accommodate heterogeneous motion, so positions whose receptive fields span regions with different true displacements

inevitably violate Eq. (4), enlarging the residual set and eroding the sparsity that delta methods rely on.

The design challenge (C1). FluxShard operates in the general regime of per-block heterogeneous MVs, where the tractability shortcut of delta methods does not apply. §**??** derives the *Receptive Field Alignment Principle*: a set of sufficient conditions, evaluated entirely on the input-level MV field, under which a layer-l output position can be certified as reusable without inspecting the neighborhoods actually assembled by warping. These conditions collapse the per-layer, per-position neighborhood check into a single resolution-aware pass over the MV field, restoring tractability while preserving correctness under arbitrary per-block motion.

Practical relaxation. The MV-inconsistent region—positions whose receptive field spans blocks with different motion vectors—is determined entirely by the spatial distribution of the MV field and does not grow across layers. However, its *relative* footprint increases sharply at deeper layers because feature map resolution decreases while the inconsistent region remains spatially fixed: a single block boundary that occupies a small fraction of the input can cover a substantial share of a low-resolution feature map. Under strict per-layer enforcement of the Receptive Field Alignment Principle, these positions are unconditionally marked for recomputation and fall outside the scope of the truncation policy, which can only reclaim positions whose delta falls below τ^l. The result is a large, non-reclaimable active set in deeper layers that dominates the computation budget.

We therefore enforce the alignment principle only at the first convolutional layer. At this layer, all inconsistent positions are recomputed exactly, eliminating the primary error from MV misalignment. In subsequent layers, these positions are treated identically to all others and governed solely by the truncation policy. This is justified by two observations. If a formerly inconsistent position produces a delta exceeding τ^l at a later layer, truncation recomputes it regardless—the relaxation has no effect. If its delta falls below τ^l, the position would have been recomputed unnecessarily under strict enforcement; skipping it introduces at most τ^l error at that layer, which is within the per-layer budget already allocated by the truncation policy. Formally, this breaks the bit-exact guarantee of Eq. **??**, but the total additional error is absorbed into the existing $\sum_l \tau^l$ budget rather than constituting an uncontrolled source. We verify empirically in §**??** that the relaxation degrades mAP by less than **XX**% while recovering **XX**% of the sparsity lost to strict enforcement.

V. Related Work

Because consecutive video frames share substantial visual content, a growing body of work caches previously computed features and reuses them for subsequent frames, reducing both computation and transmission. Existing approaches fall into two broad categories.

Pipeline-level caching. Methods such as SPINN [13] and COACH [5] cache intermediate feature maps or label-level predictions and reuse them when successive inputs are deemed globally similar. COACH, for example, maintains semantic cluster centers in the feature space and triggers an early exit—bypassing cloud inference entirely when a new frame's embedding falls within a similarity threshold. These methods make a *binary, whole-input* reuse decision: either the entire cached result is reused or it is fully recomputed. This coarse granularity suits image classification, where a single label summarizes the scene, but cannot approximate the spatially dense outputs required by segmentation or detection.

Delta-based sparse inference. A second line of work including RRM [17], CBinfer [2], Skip-Convolution [6], and DeltaCNN [19] maintains a pixel-level reference cache and propagates only the *difference* (delta) between the current frame and the reference through the network, computing only at affected output locations and reusing cached values elsewhere. Among these, DeltaCNN represents the most complete realization: it achieves end-to-end sparse propagation with truncation buffers that prevent error accumulation, enabling unbounded-length sequences without dense resets and pushing the efficiency of static-camera settings to its practical limit. MotionDeltaCNN [18] extends DeltaCNN to moving cameras by warping the cache with a single global homography before computing the delta. However, a global transformation cannot capture locally heterogeneous motion—independently moving objects, depth-induced parallax, or mixed ego-motion and object motion—leaving large residual deltas that erode sparsity.

Shared limitation. Across both categories, the cache is treated as an *indivisible, scene-level entity*: it is either globally valid, globally stale, or aligned uniformly using a single transformation. Real-world mobile video, however, exhibits motion that is local and heterogeneous: a camera pan shifts the background uniformly while foreground objects move independently, and depth discontinuities create parallax that no single warp can reconcile. MotionDeltaCNN [18] reported that their homography alignment succeeds on only a small fraction of the DAVIS [20] sequences (14 out of 80) evaluated. The result is a *granularity mismatch*: the cache granularity is the whole scene, but motion granularity is per-region. This mismatch forces existing methods to either waste computation re-deriving content that has merely shifted, or sacrifice correctness by reusing misaligned features.

Note that several other techniques also reduce the cost of video analytics, including ROI filtering [3], [14], input resolution adaptation [11], [4], frame sampling that skips inference on selected frames [3], [14], and model compression via quantization or pruning [9], [8]. These strategies are orthogonal to feature cache reuse: they govern *which* frames, regions, or model to run, whereas caching determines whether to recompute or reuse at each spatial location within a given inference. FluxShard can be composed with any of them; this work focuses exclusively on the cache reuse axis.

References

[1] Hassan J. Al Dawasari, Muhammad Bilal, Muhammad Moinuddin, Kamran Arshad, and Khaled Assaleh. Deepvision: Enhanced drone detection and recognition in visible imagery through deep learning networks. *Sensors*, 23(21), 2023.

[2] Lukas Cavigelli, Philippe Degen, and Luca Benini. CBinfer: Change-based inference for convolutional neural networks on video data.

[3] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, SenSys '15, pages 155–168. Association for Computing Machinery.
[4] Kuntai Du, Qizheng Zhang, Anton Arapin, Haodong Wang, Zhengxu Xia, and Junchen Jiang. AccMPEG: Optimizing video encoding for video analytics.
[5] Luyao Gao, Jianchun Liu, Hongli Xu, Sun Xu, Qianpiao Ma, and Liusheng Huang. Accelerating end-cloud collaborative inference via near bubble-free pipeline optimization. *arXiv preprint arXiv:2501.12388*, 2024.
[6] Amirhossein Habibian, Davide Abati, Taco S. Cohen, and Babak Ehteshami Bejnordi. Skip-convolutions for efficient video processing. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2694–2703. ISSN: 2575-7075.
[7] Beining Han, Meenal Parakh, Derek Geng, Jack A. Defay, Gan Luyang, and Jia Deng. FetchBench: A simulation benchmark for robot fetching.
[8] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 1*, volume 1 of *NIPS'15*, pages 1135–1143. MIT Press.
[9] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2704–2713. IEEE.
[10] Fatemeh Jalali, Morteza Khademi, Abbas Ebrahimi Moghadam, and Hadi Sadoghi Yazdi. Robust scene aware multi-object tracking for surveillance videos. 638:130114.
[11] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 253–266. Association for Computing Machinery.
[12] Rahima Khanam and Muhammad Hussain. Yolov11: An overview of the key architectural enhancements. *arXiv preprint arXiv:2410.17725*, 2024.
[13] Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D Lane. Spinn: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th annual international conference on mobile computing and networking*, pages 1–15, 2020.
[14] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. Reducto: On-camera filtering for resource-efficient real-time video analytics. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '20, pages 359–376. Association for Computing Machinery.
[15] Mostafa Mahmoud, Kevin Siu, and Andreas Moshovos. Diffy: a déjà vu-free differential deep neural network accelerator. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-51, pages 134–147. IEEE Press.
[16] Kien Nguyen, Feng Liu, Clinton Fookes, Sridha Sridharan, Xiaoming Liu, and Arun Ross. Person recognition in aerial surveillance: A decade survey. pages 1–1.
[17] Bowen Pan, Wuwei Lin, Xiaolin Fang, Chaoqin Huang, Bolei Zhou, and Cewu Lu. Recurrent residual module for fast inference in videos. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1536–1545. ISSN: 2575-7075.
[18] Mathias Parger, Chengcheng Tang, Thomas Neff, Christopher D. Twigg, Cem Keskin, Robert Wang, and Markus Steinberger. MotionDeltaCNN: Sparse CNN inference of frame differences in moving camera videos with spherical buffers and padded convolutions. pages 17292–17301.
[19] Mathias Parger, Chengcheng Tang, Christopher D Twigg, Cem Keskin, Robert Wang, and Markus Steinberger. Deltacnn: End-to-end cnn inference of sparse frame differences in videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12497–12506, 2022.
[20] Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alexander Sorkine-Hornung, and Luc Van Gool. The 2017 davis challenge on video object segmentation. *arXiv:1704.00675*, 2017.
[21] Timo von Marcard, Roberto Henschel, Michael Black, Bodo Rosenhahn, and Gerard Pons-Moll. Recovering accurate 3d human pose in the wild using imus and a moving camera. In *European Conference on Computer Vision (ECCV)*, sep 2018.
[22] Qi Wu, Zipeng Fu, Xuxin Cheng, Xiaolong Wang, and Chelsea Finn. Helpful DoggyBot: Open-world object fetching using legged robots and vision-language models.
[23] Jun Zhang, Mina Henein, Robert Mahony, and Viorela Ila. VDO-SLAM: A visual dynamic object-aware SLAM system.

A. Biographies and Author Photos

```
\begin{IEEEbiographynophoto}{Jane Doe}
Biography text here without a photo.
\end{IEEEbiographynophoto}
```

or a biography with a photo

```
\begin{IEEEbiography}[{\includegraphics
[width=1in,height=1.25in,clip,
keepaspectratio]{fig1.png}}]
{IEEE Publications Technology Team}
In this paragraph you can place
your educational, professional background
and research and other interests.
\end{IEEEbiography}
```

Please see the end of this document to see the output of these coding examples.

Jane Doe Biography text here without a photo.

IEEE Publications Technology Team In this paragraph you can place your educational, professional background and research and other interests.