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

IEEE Publication Technology Department

***Abstract*—In edge-cloud video analytics, limited edge computation power and bandwidth make feature reuse essential to sustain accuracy under strict latency constraints. Prior reuse strategies often fail in dynamic scenes where motion and viewpoint changes disrupt naive content matching and reusing. We formulate this challenge as a *motion-induced cache-remapping problem* and design a lightweight motion-vector-guided abstraction to align reusable features and identify true misses requiring recomputation. Around this abstraction, we propose FluxShard, a motion-aware collaborative video analytics system that prioritizes task-relevant updates, dispatches inference workload to edge and cloud adaptively to the dynamic network conditions, and maintains feature freshness without stalling inference. In the way, FluxShard accelerates both computation and communication. Evaluation shows that FluxShard achieves 40% to 85% bandwidth reduction, 2.3× to 5.5× speedup, and ∼99% accuracy retention on various video analytics tasks across different datasets, compared to state-of-the-art baselines. This formulation and system provide a principled foundation for motion-aware reuse in resource-constrained video analytics.**

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

I. Introduction

Video analytics underpins a wide range of latency-critical applications such as embodied intelligence [50], [10], aerial drones [1], [32], augmented reality [53], and smart surveillance [16], where timely and accurate understanding of high-rate video streams is essential. Processing solely on edge devices erases transmission latency but is constrained by limited computational and energy resources. Conversely, offloading to the cloud offers abundant computing power but introduces significant network latency, undermining real-time performance. A promising way to mitigate both transmission and computation costs is to ***reuse*** results across consecutive frames in a cache-like manner: if parts of the scene remain unchanged, their previously computed outputs can be retained instead of recomputed or re-sent, reducing both processing delay and bandwidth usage while keeping results consistent.

However, real-world video streams are rarely static. Camera motion, object dynamics, and viewpoint shifts cause scene content to displace spatially across frames. Such displacements break naive pixel-wise or block-wise matching, leading to frequent cache misses and forcing redundant computation or transmission. These inefficiencies undermine the potential latency and bandwidth savings of cache-like reuse, motivating the need for more robust mechanisms that can tolerate motion while preserving accuracy.

Prior work has aimed to reduce redundant computation through cache-like reuse at the granularity of the ***whole scene***, a strategy effective in static scenes but fundamentally challenged by various types of motion in dynamic environments: for consecutive video frames, 1. SPINN [20] and COACH [9] cache and reuse the whole intermediate feature maps for similar frames; 2. CBinfer [4], Diffy [24] and DeltaCNN [35] maintain a master cache and update it with incremental pixel-wise changes; 3. MotionDeltaCNN [34] attempts to accommodate camera ego-motion by rolling the master cache indices via a global homography matrix, a technique largely restricted to parallel translations and planar targets. The success of these paradigms hinges on a core assumption of spatio-temporal consistency of the whole scene, which is frequently violated by various types of motion in dynamic environments. This exposes the central challenge for systems operating in such scenarios: to maintain feature consistency efficiently through complex motion.

A promising direction for resolving this challenge is to incorporate motion vectors (MVs), a concept long used in video encoding to capture ***block-level*** displacement between consecutive frames. Such information reflects appearance-level similarity over time and can help track the movement of reusable content, thereby reducing false cache misses that arise when shifts or viewpoint changes are misinterpreted as content changes. However, motion vectors have inherent limits: as block-based approximations, they fail to model complex transformations such as non-rigid deformations, scaling, or occlusions. Consequently, feature misalignment persists and recomputation is necessary. Here, we take the concept of motion vectors out of the heavy encoding pipeline and use it efficiently as a lightweight, model-agnostic signal both to guide cache alignment for reuse and to identify regions that require recomputation in dynamic video analytics.

We present ***FluxShard***, a motion-aware system that optimizes end-to-end latency of edge-cloud video analytics through fine-grained, motion-guided feature alignment. FluxShard uses motion vectors (MVs) as a lightweight control signal to align cached features and isolate minimal recomputation regions. By quantifying the resulting sparse workload against network conditions, FluxShard dynamically selects the optimal execution path—either local sparse updates or cloud offloading of non-reusable updates. This co-optimization of transmission and computation enables low-latency inference even in highly dynamic environments.

The design of FluxShard has the following challenges. The first challenge is maintaining semantic consistency in layers with spatial dependencies (e.g., convolutions with kernel size larger than 1). Naive spatial alignment of cached features fails when an output pixel depends on a neighborhood that

has moved inconsistently. We propose a **General Receptive-Field Alignment Principle** which defines the necessary and sufficient conditions for valid reuse regions and guides recomputation. This principle ensures that the fused feature maps remain bit-equivalent to full-frame inference, preventing the accumulation of spatial artifacts across deep layers.

The second challenge is sparsity erosion, where sparse updates "bleed" and expand across cascaded layers, diminishing performance gains. FluxShard suppresses this via a profiling-driven truncation policy: we offline characterize the optimal mapping between inter-frame motion statistics (e.g., MV variance) and the tightest accuracy-preserving thresholds. At runtime, the system dynamically modulates truncation based on this pre-defined mapping. This effectively halts redundant activation propagation with negligible decision overhead, maintaining high sparsity even in dynamic scenes.

We implement FluxShard by developing a comprehensive suite of motion-aligned CNN operators, building upon and significantly extending the DeltaCNN framework. Specifically, we re-engineered DeltaCNN's high-performance convolution kernels and integrated custom-built Triton kernels to optimize the irregular memory patterns inherent in MV-guided sparse workloads. We evaluate FluxShard on two representative real-world video analytics tasks: object detection and object segmentation for high-resolution environments using YOLOv11 [19]. The baselines include Naive Offloading, COACH [9], DeltaCNN [35] and MotionDeltaCNN [34], tested on NVIDIA Jetson Xavier NX devices (edge) and an RTX 3080 PC (cloud). Across all tasks and datasets, FluxShard achieves:

- **Bandwidth reduction** - up to **92%** savings over full-frame transmission. TODO: change to xx to xx%
- **Energy efficiency** - up to **xx%** savings over full-frame transmission. TODO: change to xx to xx%
- **End-to-end acceleration** - up to **5.5**× faster inference compared to the baselines. TODO: change to xx to xx%
- **Accuracy retention** - preserves up to **99%** of peak accuracy. TODO: change to xx to xx%
- **Scalability** - maintains ∼xx% lower latency growth than baselines when scaling to multiple concurrent edge devices.

In summary, FluxShard provides a holistic solution for high-performance edge-cloud video analytics in dynamic environments by harmonizing theoretical principles with low-level systems optimization. Our work contributes a General Receptive-Field Alignment Principle to ensure bit-equivalent semantic consistency, coupled with a profiling-driven truncation policy that adaptively maintains execution efficiency under high dynamics. We realize these innovations through a unified motion-aligned operator library that bridges the gap between theoretical reuse principles and hardware-efficient sparse execution. Extensive evaluations demonstrate that FluxShard achieves significant reductions in both bandwidth and end-to-end latency while maintaining near-peak accuracy, offering a robust framework for the next generation of latency-critical collaborative intelligence.

II. Background

Real-time video analytics has become a cornerstone of modern intelligent systems, powering applications such as autonomous vehicles [28], [29], [30], augmented reality [42], [37], [8], video surveillance [5], [14], [45], and smart city infrastructure [2], [3], [13]. These applications often require immediate responses based on the continuous analysis of high-resolution video streams, which generate immense computational and bandwidth demands. While cloud computing offers powerful resources for processing such workloads, the latency and reliability factors of remote data transmission can introduce significant delays. Conversely, edge devices, although closer to the data source, are typically resource-constrained, with limited computational power, memory, and energy efficiency.

A. Main Models in Visual Analytics

Visual analytics tasks, such as object detection [33], [21], [51], semantic segmentation [41], [40], [23], optical flow estimation [43], [44], [15], and depth estimation [18], [26], [27], rely on advanced deep learning models to extract fine-grained spatial information from video data. These models predominantly fall into two families: convolutional neural networks (CNNs) and vision transformers.

1) Spatial Computation in CNNs and Vision Transformers: CNNs are inherently spatially structured, with convolutional operations preserving the spatial relationships within feature maps. For example, models like Faster R-CNN [46] and DeepLab [6] retain pixel-to-pixel alignment between input frames and feature maps during each convolution and pooling operation, enabling precise spatial reasoning in tasks like segmentation or detection.

Vision transformers, such as DETR [54] and SegFormer [52], divide input frames into spatial patches before applying self-attention mechanisms. While transformers lack the strict locality bias of CNNs, they still maintain patch-wise spatial alignment throughout their computation pipeline, which we can exploit to track regions of interest.

2) Motion Vector-Wrapped Blocks in CNNs and Transformers: FluxShard leverages the spatial alignment preserved by both CNNs and vision transformers to enable its motion vector-wrapped block abstraction. For CNNs, the blocked structure seamlessly integrates with the convolutional operations, as each block intrinsically corresponds to a subset of the spatially aligned feature map. Motion-sensitive updates can be computed efficiently by isolating block-level regions and propagating changes through the subsequent layers.

For vision transformers, the motion vector-wrapped block abstraction aligns naturally with the patch-based input representation. Dynamic patches are tracked and updated using motion vectors, enabling FluxShard to focus attention computations and self-attention mechanisms on only the changing regions of the frame. This enables efficient sparse processing without disrupting the global-context modeling characteristic of transformers.

B. Challenges in Edge-Cloud Video Analytics

Edge-cloud architectures must balance computation, bandwidth, and latency, but stringent resource constraints hinder real-time video analytics.

Bandwidth Bottlenecks: High-definition video streaming (e.g., 1080p @ 30 FPS) demands substantial uplink bandwidth, even with H.264 compression—ranging from 8˜20 Mbps [47]. Multi-camera setups exacerbate this issue, with 10-camera systems requiring up to 200 Mbps, far exceeding practical limits in mobile networks, where uplink bandwidth often falls below 10 Mbps.

Edge Computation Limits: Resource-constrained edge devices struggle with real-time neural inference. For instance, YOLOv11 achieves only 5 FPS on a etson Xavier NX in our evaluation, operating near peak GPU utilization. Energy constraints in battery-powered devices further limit their capacity to sustain continuous analytics.

Latency Sensitivity: Autonomous driving and surveillance demand perception latencies below 50˜100 ms [22]. However, cloud offloading introduces 50˜200 ms round-trip delays [25], making such deadlines difficult to achieve.

To address these constraints, FluxShard introduces **motion vector-wrapped blocks**, reducing transmission overhead by prioritizing motion-sensitive updates while dynamically balancing edge-cloud computing to minimize latency.

C. Existing Systems and Their Limitations

Several frameworks have been proposed to tackle edge-cloud collaboration for video analytics, but they exhibit fundamental shortcomings in harnessing the spatiotemporal redundancy of video streams and addressing dynamic scene changes.

Cache-pipeline-based systems, such as SPINN [20] and COACH [9], decompose video analytics into stages like frame sampling, feature extraction, and inference, distributing computation between the edge and cloud. During inference, they attempt to reuse computation by caching early-exit results. While effective for simpler recognition tasks (e.g., classification), they struggle with dense workloads like segmentation or keypoint detection, which require precise spatial details.

As a result, inference results cannot be directly reused or approximated, necessitating the transmission of full or compressed intermediate data (e.g., feature maps or video frames) to the cloud, leading to:

- *High Transmission Overhead:* Sending entire feature maps or frames strains bandwidth, limiting scalability in multi-camera setups.
- *Redundant Processing:* These methods lack motion-awareness and recompute both static and dynamic regions, increasing computational overhead.
- *Accuracy Degradation:* Even when frame-to-frame similarity exceeds 95%, the critical changes often occur within the 5% non-similar regions, which are essential for dense tasks. Cached results fail to capture these fine-grained updates, producing coarse segment boundaries in object segmentation and unstable keypoint estimates in pose detection, degrading accuracy.

Delta-based systems, such as DeltaCNN [35], improve upon pipeline-based methods by exploiting the temporal redundancy of video streams. These methods transmit and process only the changed regions (*deltas*) between consecutive frames. While this reduces data transmission and computation, delta-based approaches face critical limitations:

- *State Inconsistency:* Without explicit consideration of motion vectors, deltas fail to account for global scene changes, such as shifts caused by camera motion, leading to redundant updates and loss of context.
- *Inefficient Sparse Computation:* Sparse updates introduce irregular memory access patterns that degrade the performance of GPU-based dense computation kernels, limiting efficiency gains.

D. Towards a Motion-Aware Abstraction

Despite progress in cache-pipeline-based and delta-based current systems fail to address the unique challenges posed by highly dynamic, resource-constrained scenarios of dense visual analytics. This motivates a new abstraction that:

- **Encodes Spatiotemporal Redundancy:** Explicitly identifies and processes only localized, dynamic regions of video streams over time.
- **Scales Across Motion Dynamics:** Maintains state consistency while adapting to global scene shifts using motion-aware information.
- **Enables Unified Optimization:** Integrates computation, communication, and state propagation to holistically optimize bandwidth, accuracy, and latency.

In this work, we introduce the **motion vector-wrapped block abstraction** as the foundational design principle for **FluxShard**, a framework that addresses these challenges and sets a new paradigm for edge-cloud collaborative video analytics.

E. Relationship With Motion-Vector-Based Video Compression Standards

Modern video compression standards such as H.264 [49], H.265/HEVC [36], VP9 [31], and AV1 [11] reduce bandwidth requirements by exploiting redundancies in video streams through motion-vector-based interframe compression. These methods analyze temporal changes between consecutive frames, using motion vectors to represent the displacement of regions, thereby minimizing redundant data transmission.

FluxShard builds upon this principle by reusing motion vectors from these video codecs to guide its inference optimization pipeline. Specifically, the motion vector-wrapped block abstraction in FluxShard leverages these motion vectors to identify and update dynamically changing regions, avoiding unnecessary computation on unchanged spatial regions. This enables FluxShard to minimize the computational and communication overheads for video analytics tasks.

While motion-vector-based codecs optimize video encoding for efficient storage or transmission, FluxShard complements this by targeting computation and bandwidth efficiency at the inference level, where feature extraction and model processing

dominate. By unifying these layers in the video analytics pipeline, FluxShard ensures system-wide efficiency in both video delivery and AI-driven analysis workflows.

III. Overview

FluxShard is designed for real-time video analytics in resource-constrained edge-cloud settings, such as drones, robots, or smart cameras, where high-resolution video must be processed under both computation limits and fluctuating wireless bandwidth. The key idea is to maintain *motion-induced cache coherence* across devices: coherence here does not mean keeping edge and cloud caches identical at every step, but rather ensuring that with motion warping and sparse updates, the features required for the current frame form a coherent view on at least one side. Intermediate features are opportunistically reused whenever motion allows, while true misses are selectively recomputed by either the edge or the cloud. Figure 1 illustrates the design.

A. Key Components

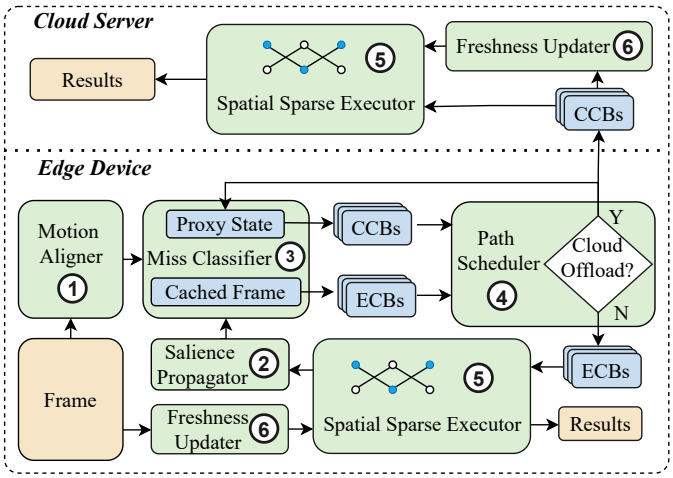

Figure 1. Workflow of FluxShard.

FluxShard coordinates several lightweight components to realize this design:

1. Motion Aligner. Estimates block-level motion between frames and warps cached features on edge and cloud so that large regions can be reused instead of recomputed.

2. Salience Propagator. Projects activation salience from the previous frame into the current one using the motion field, highlighting accuracy-sensitive regions.

3. Miss Classifier. Consolidates motion and salience signals into a set of *critical blocks* that must be refreshed to sustain coherence with the current input frame. As part of this classifier, we embed a lightweight proxy state that tracks the server's current reference image by bending the sent critical blocks into the reference image. The classifier ensures that, after motion warping and selective updates, the cached features on each side form a representation consistent with the frame. Since edge and cloud differ in compute capacity and cache freshness, their motion vectors and respective critical sets may not be identical and thus we have edge critical blocks (ECBs) and cloud critical blocks (CCBs).

4. Path Scheduler. With the salience-guided critical blocks identified and the current estimated network bandwidth, the scheduler decides whether edge or cloud should execute them for the current frame. The decision reflects available compute and bandwidth so that inference can be completed in a timely manner while maintaining coherence with the input. The chosen side produces the result, while the other may carry on background updates that improve cache freshness for future frames.

5. Spatial Sparse Executor. Executes the identified critical blocks with motion vector-guided indexing. In video analytics models where convolutions dominate, sparse execution is complicated by the need for neighborhood context (due to kernel padding) and by cache misalignment under frame-level motion. The executor leverages motion vectors to guide indexing, so that each block is augmented with only the boundary elements it depends on, while cached features are still correctly referenced even when global shifts occur. This motion-aware indexing ensures that sparse execution remains efficient and cache hits remain valid under motion, complementing the upstream coherence mechanisms.

6. Freshness Updater. Runs dense computation opportunistically when slack is available, refreshing stale cache regions to increase the chance of coherence in future frames.

B. System Workflow

At system startup, the edge and cloud share a synchronized reference frame along with a consistent set of cached features. This establishes a common starting point for subsequent collaborative inference.

Given a new frame, the *Motion Aligner* and *Salience Propagator* highlight blocks likely to miss, which the *Miss Classifier* consolidates into a coherent set of critical blocks for both ends. The *Path Scheduler* then evaluates both edge and cloud execution feasibility using startup compute profiles and real-time bandwidth measurements and selects one side to process this set. The chosen side invokes the *Spatial Sparse Executor* to compute the blocks efficiently and produces the final inference result. If any CCBs are transmitted, *Proxy State* for the server is incrementally synchronized to closely track the server's current reference image.

In parallel, the system opportunistically maintains cross-end consistency: while one side is occupied with critical block execution, bandwidth gaps are leveraged to incrementally synchronize *Proxy State*, and idle compute slots are utilized by the *Freshness Updater* to refresh cached features. These overlapping background actions ensure that both edge and cloud remain closely aligned for future frames at minimal extra cost.

Through this workflow, FluxShard enforces motion-induced coherence at the level of a full inference while still exploiting fine-grained sparsity. This enables accurate, low-latency video analytics under tight edge-cloud resource and bandwidth constraints.

IV. Design

Building on the aforementioned components, this section details their concrete design and interactions, including how motion fields align cached features, how salience guides block selection, how misses are consolidated and scheduled, how sparse execution preserves efficiency under motion, and how background updates refresh cache freshness over time, thereby enabling FluxShard to achieve motion-induced coherence for accurate and low-latency video analytics.

A. Motion Aligner

To enable temporal reuse of features, we first compensate for local displacements caused by object or camera motion. We employ a block-based motion alignment that estimates motion vectors between consecutive frames via block matching. For each block $\Omega_{x,y}$ in frame t, the algorithm searches candidate displacements $(\Delta x, \Delta y)$ within a local window $\mathcal{W}$ in the previous frame $t-1$. Each candidate is scored by the sum of absolute differences (SAD):

$$\mathrm{SAD}_{x,y}(\Delta x, \Delta y) = \sum_{(u,v)\in\Omega_{x,y}} \big| I_t(u,v) - I_{t-1}(u+\Delta x,\, v+\Delta y) \big|.$$

The displacement with the lowest score is selected as the motion vector:

$$(\Delta x^*, \Delta y^*) = \arg\min_{(\Delta x,\Delta y)\in\mathcal{W}} \mathrm{SAD}_{x,y}(\Delta x, \Delta y).$$

By construction, this procedure always provides the best-matching offset within the search window, and when the minimum cost is sufficiently small, the corresponding aligned features are reliable for reuse. However, in regions subject to strong motion, occlusion, or lack of texture, the minimal SAD may still exceed a predefined threshold τ, indicating that even the "best" candidate is in fact poorly aligned. We treat such cases as *true misses*: although a motion vector can always be produced, it should not be trusted for propagation. Instead, blocks with matching cost $\mathrm{SAD}_{x,y}(\Delta x^*, \Delta y^*) > \tau$ are marked as *recomputation candidates*, to be revisited in the subsequent salience analysis.

B. Salience Propagator

Although all high-cost blocks from motion alignment are marked as recomputation candidates, not every candidate influences the final task prediction equally. To prioritize compute for the most relevant regions, we introduce a *salience propagator* that exploits task-level cues to estimate importance.

Instead of relying on heavyweight attention mechanisms, the salience score $s_{x,y}$ for each candidate block (x, y) is computed in a lightweight, task-aware manner by measuring its overlap with task-driven priors (e.g., segmentation masks, detected objects, or other confidence maps) derived from previous outputs. This provides a direct proxy of which regions are semantically crucial for the downstream task. Formally, let $\mathcal{C}$ denote the set of recomputation candidates identified by the motion aligner. Each $(x, y) \in \mathcal{C}$ is assigned a salience score $s_{x,y} \in [0, 1]$. A block is regarded as *salient* if

$$s_{x,y} \;\geq\; \sigma,$$

where σ is a task-driven threshold. Otherwise, the block is classified as *non-salient*. This filtering step ensures that recomputation is reserved only for regions likely to impact final predictions, while less critical areas can be approximated by aligned features from cached frames.

C. Miss Classifier

Given the motion cost $\mathrm{SAD}_{x,y}(\Delta x^*, \Delta y^*)$ from the aligner and the salience score $s_{x,y}$ from the propagator, the miss classifier decides whether block (x, y) should be recomputed or can be approximated by motion-aligned reuse.

The decision rule is summarized as:

$$F'_{x,y} = \begin{cases} \mathrm{Aligned}(F_{t-1}, M_{x,y}), & \text{if } \mathrm{SAD}_{x,y} \leq \tau \;\text{ or }\; s_{x,y} < \sigma, \\ \mathrm{Recompute}(I_t), & \text{if } \mathrm{SAD}_{x,y} > \tau \;\text{ and }\; s_{x,y} \geq \sigma, \end{cases}$$

where $M_{x,y}$ is the estimated motion vector, I_t is the current frame, τ controls alignment reliability, and σ is the salience threshold. We empirically set $\tau = 0.025$ and σ is task dependent.

Two groups of critical blocks. Since the reference features available at the edge and at the server differ, applying the same rule produces two corresponding sets of recomputation requirements: 1. *Edge Critical Blocks (ECBs)*, determined using edge-side references and their motion vectors; 2. *Cloud Critical Blocks (CCBs)*, determined using server-side references (also maintained by the edge as the proxy state) and their motion vectors.

Each set ensures that inference at its respective endpoint maintains temporal coherence to the current frame. With both critical sets established, the next step is to decide which side should carry out the necessary recomputation in order to achieve the best end-to-end efficiency.

D. Path Scheduler

We assume that execution time under different environments can be profiled as functions of the number of recomputed blocks: $f_e(\cdot)$ on the edge and $f_c(\cdot)$ on the cloud. For transmission overhead, each block has average size s, so sending N_{CCB} critical blocks takes $\frac{N_{\mathrm{CCB}} \cdot s}{B}$ time under bandwidth B. Hence

$$T_{\mathrm{edge}} = f_e(N_{\mathrm{ECB}}), \qquad T_{\mathrm{cloud}} = f_c(N_{\mathrm{CCB}}) + \frac{N_{\mathrm{CCB}} \cdot s}{B}.$$

The scheduler then selects the path with the smaller cost,

$$\arg\min_{p\in\{\mathrm{edge},\,\mathrm{cloud}\}} T_p,$$

thereby guaranteeing minimal inference latency without any compromise to the accuracy of the baseline model

E. Spatial Sparse Executor

The spatial sparse executor bridges block scheduling and actual computation (Fig. 2). In the previous stage, the scheduler marks *critical blocks* that require recomputation. These blocks are always extracted directly from the current input image, and thus their central regions do not rely on motion alignment. This

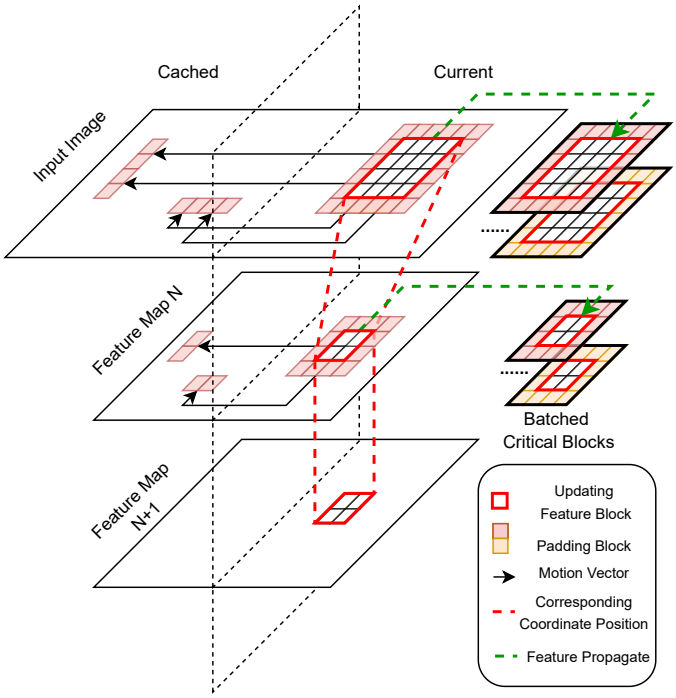

Figure 2. Demonstration of the intermediate propagation of the spatial sparse executor. The red line shows the coordinate correpondance relationship and the green line shows the actual data flow.

differs from propagated blocks, whose contents are carried from past frames via motion-guided warping.

To guarantee consistency with dense inference, however, each critical block must be expanded with surrounding padding to recover the full receptive field. Here motion offsets become indispensable: while the block centers come from the current image, their border context would be mismatched without motion alignment. We therefore employ *offset-aware padding*: the main block region is taken verbatim from the current frame, while the padding is sampled according to motion vectors. Offsets are discretized to the nearest neighbor, which stabilizes alignment and avoids the boundary erosion often caused by bilinear schemes.

During intermediate propagation, all reused and recomputed blocks remain in this block-wise format with offset-aware padding. We intentionally avoid forcing a full-frame alignment at every layer, since such global overwriting would convert sparsity back into dense computation and undermine efficiency. Instead, blocks are concatenated along the batch dimension so that irregular, motion-guided updates can be executed as uniform batched workloads.

Finally, at the *model output stage*, the executor performs one global alignment and update. All blocks are projected back to their designated global coordinates, and recomputed critical blocks overwrite outdated content. This final merging step restores dense-level consistency only at the prediction boundary, balancing efficiency during intermediate propagation with accuracy at the output.

F. Freshness Updater

Block-sparse execution ensures that each frame can be processed within a tight latency budget, but over time the cache may drift away from what would have been produced by dense inference. This gradually weakens the coherence between cached representations and the visual content of the current input. To counteract such drift without sacrificing responsiveness, we exploit opportunities in both computation and communication, allowing accuracy to recover opportunistically while the sparse critical path remains intact.

On the client side, whenever cycles become available, the system performs small fragments of a dense update. These updates are preemptable: they run only when compute resources are idle, and yield immediately when the next sparse inference step is scheduled. Incrementally, they rewrite parts of the cache with dense features, tightening its coherence with the live input stream and reducing long-term degradation under reuse.

Communication offers a complementary opportunity. The server strictly requires only critical blocks, which keeps baseline bandwidth low. However, whenever residual capacity is available, the client can forward additional non-critical blocks. These opportunistic transmissions gradually align the server-side cache with the client's current frame, improving feature coherence across devices without imposing extra latency.

Through this dual mechanism, spare compute cycles and residual bandwidth are repurposed to continually reinforce the model state. The result is a system where caches remain coherent both locally and remotely, progressively approaching the fidelity of dense execution while retaining the efficiency of the sparse pipeline.

V. Overall Workflow

Putting the above components together, FluxShard processes each frame through the cooperative workflow summarized in Algorithm 1. At the edge, the system derives motion vectors, salience, and criticality from the cached previous frame (together with the proxy state of). Based on current bandwidth and compute capacities, it compares the estimated latency of local execution with that of offloading, and dynamically assigns the critical path to either edge or server. When the edge executes locally, it opportunistically transmits spare non-critical blocks to the server; when the server takes over, the edge instead refreshes stale cache regions. The server, in both cases, integrates any received non-critical blocks into its cache. Through this cooperative mechanism, FluxShard achieves low-latency inference while keeping cache states consistent across edge and server. The coherence maintained between motion alignment, salience propagation, and opportunistic updates ensures that both accuracy and efficiency are sustained throughout continuous video processing.

VI. Implementation

We prototype *FluxShard* on Ubuntu 20.04 with Python and C++/CUDA. The core computational modules—including motion alignment, salience scoring, and miss classification—are packaged as a custom *PyTorch extension*. This fused design avoids unnecessary host-device transfers: motion search, cost

Algorithm 1: Cooperative execution of FluxShard at frame F_t

Input: Frame F_t, previous cached frame F_{t-1}, proxy state for server P_{t-1}, bandwidth B_t
Output: Inference result Y_t

Edge-side procedure: ;
  $(\mathcal{M}_e, \mathcal{M}_s, SAD_e, SAD_s) \leftarrow$ MotionAligner(F_{t-1}, P_{t-1}, F_t) ;
  $S_t \leftarrow$ SaliencePropagator$(F_{t-1}, \mathcal{M}_e, \mathcal{M}_s)$;
  $(\mathcal{B}_e, \mathcal{B}_c) \leftarrow$ MissClassifier(S_t, SAD_e, SAD_s) ;
  $T_e \leftarrow$ estimate_edge_time$(\mathcal{B}_e)$;
  $T_c \leftarrow$ estimate_cloud_time$(\mathcal{B}_c, B_t)$;
  if $T_e \leq T_c$ **then**
    /* Edge executes the critical path */
    $Y_t \leftarrow$ SparseExecutor$_{edge}(\mathcal{B}_e, \mathcal{M}_e)$;
    Opportunistically transmit extra non-critical blocks to server ;
    Update proxy state P_t with sent non-critical blocks ;
  else
    /* Server executes the critical path */
    Send $\mathcal{B}_c$ and $\mathcal{M}_c$ to server ;
    Opportunistically refresh stale blocks via freshness updater ;
    Update proxy state P_t with $\mathcal{B}_c$ and $\mathcal{M}_c$;

Server-side procedure: ;
  if *received* $\mathcal{B}_c$ *and* $\mathcal{M}_e$ **then**
    $Y_t \leftarrow$ SparseExecutor$_{server}(\mathcal{B}_c, \mathcal{M}_e)$;
  Opportunistically integrate received non-critical blocks into cache ;

return Y_t

aggregation, and block selection execute directly as CUDA kernels with shared buffers, and results are exposed as PyTorch tensors for seamless integration with subsequent model layers.

To maintain long-term coherence, we implement the *freshness updater* as a Python generator. All model operators are profiled into a callable sequence; the updater iterates via a yielded `for`-loop, allowing background refresh computations to pause and resume at operator boundaries. This ensures that opportunistic updates exploit only idle cycles without interfering with latency-critical inference. Complementarily, the updater also leverages residual bandwidth to gradually transmit non-critical blocks, aligning server-side caches at negligible cost.

Edge-cloud communication is realized via a minimal TCP runtime with block-structured serialization. Despite its simplicity, this lightweight design sustains real-time throughput over wireless links while keeping the codebase portable across heterogeneous devices.

VII. Evaluation

A. Evaluation Setup

a) Testbed.: We evaluate FluxShard on a hybrid edge-cloud testbed comprising several NVIDIA Jetson Xavier NX devices (384 CUDA cores, 8 GB LPDDR4) and a cloud server with an NVIDIA RTX 3080 GPU (10 GB GDDR6X). Both run Ubuntu 20.04 with CUDA 11. Edge devices handle preprocessing and local inference, while the cloud provides additional compute resources for offloaded tasks. All devices connect via a 1000 Mbps Ethernet switch.

To simulate real-world LTE/5G networks, we apply bandwidth-limited traces from the Madrid LTE Dataset [39] via the Linux `tc` utility, enforcing bandwidth and latency constraints:

- High (130 Mbps): Optimal LTE/5G conditions.
- Medium (56 Mbps): Moderately loaded networks.
- Low (25 Mbps): Congested or degraded conditions.

b) Models and Datasets.: We evaluate FluxShard mainly on two most common video analytics deep tasks including object detection using YOLO11m [17] (referred to as Detect) and dense segmentation using YOLO11m-seg [17] (referred to as Segment) on two real-world datasets: 3DPW [48] and DAVIS [38]. We use the medium version of their family of models to balance between the accuracy and the latency on the edge. DAVIS is a video dataset featuring a large variety of moving object categories in the wild and diverse camera motion, while 3DPW hosts video sequences of human activities in cities captured from a mildly moving hand-held camera. We also include two extra tasks to demonstrate the further impact of the application of FluxShard on the other tasks: image classification using Vision Transformer [7] (referred to as Classify) and multi-person keypoint detection using Mask R-CNN [12] (referred to as Keypoint).

The statistics for Detect and Segment and their target datasets are shown in Table I.

Table I
STATISTICS OF THE EVALUATED TASKS AND MODELS.

Model	Detect	Segment
Dataset	3DPW	DAVIS
Resolution	1024×1024	1024×1024
Parameter (M)	20.1	22.4
Server Latency (ms)	17.56	22.23
Edge Latency (ms)	386.26	619.82
Edge Power (mW)	12696.7	15736.5

c) Baselines: We compare FluxShard against four baselines:

- Full Offload: Executes all inference on the server, incurring high transmission costs.
- COACH [9]: Pipeline-based method with early exit for computation reduction (by comparing the similarity of the current frame with the cached history frames) and quantization for bandwidth reduction.
- DeltaCNN [35]: Processes only pixel-level frame deltas.
- ROI: Vanilla ROI-based method. Processes only the regions of interest (ROI) of the frame, which is the bound-

ing boxes of the detected objects by a small yolo11n-detect model [17] with only 2.6M parameters on the edge.

d) Metrics.: FluxShard's evaluation considers:

- Accuracy: Intersection over Union (IoU) for Detect and Segment, top-1 accuracy for Classify and mean Average Precision (mAP) for Keypoint, normalized by the accuracy of the larger version of the model (for the YOLO series of models) or the full model (for the others).
- Latency: Average end-to-end frame processing time on the critical path.
- Bandwidth Usage: Average transmission bandwidth (MBps).
- Cache Hit Rate: Cache reuse ratio under different bandwidth conditions and different scene motions; for COACH, it measures early-exit computation reuse and for ROI, it measures the transmission saved by only processing the regions of interest.
- Compute Load Distribution: Fraction of total computations on the critical path handled on edge and cloud.

B. End-to-End Results with a Single Edge Device

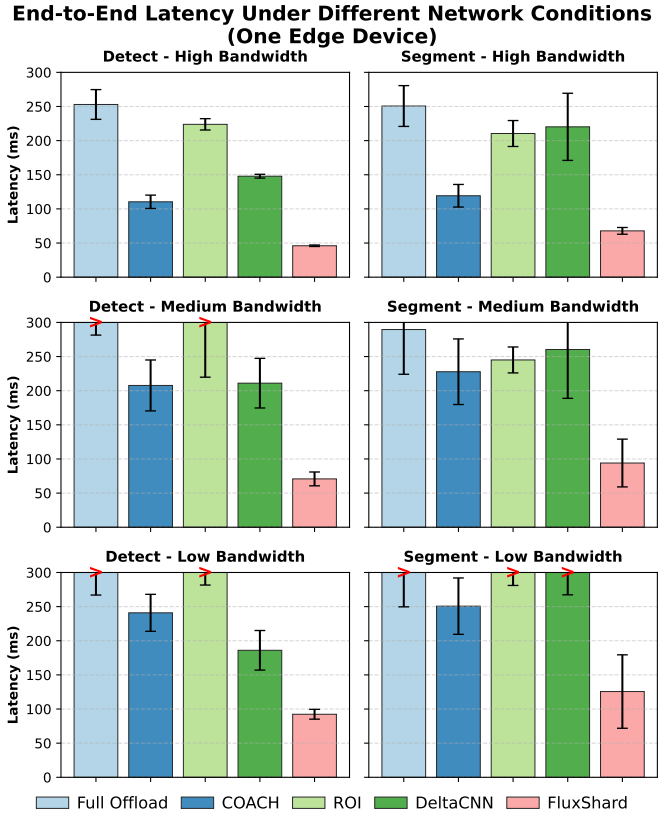

Figure 3. Latency comparison of different systems. FluxShard consistently outperforms the other baselines under different bandwidth conditions.

a) End-to-End Latency: Figure 3 plots the end-to-end latency across bandwidth levels, showing FluxShard as consistently the fastest method. In high-bandwidth Detect, FluxShard shortens execution by about $5.5\times$ compared to Full Offload, $2.4\times$ against COACH, $3.2\times$ against DeltaCNN, and nearly $5\times$ relative to ROI. For segmentation, the gains remain strong with $3.7\times$ over Full Offload and 1.8–$3.2\times$ over other baselines. At medium bandwidth, FluxShard maintains substantial advantages: $4.5\times$ faster than Full Offload in Detect, 2.9–$3.0\times$ over COACH and DeltaCNN, and $4.4\times$ relative to ROI. For segmentation, the margins are $3.1\times$ over Full Offload and 2.4–$2.8\times$ against COACH and DeltaCNN, with $2.6\times$ speedup over ROI. Even under low bandwidth—the most challenging regime—the improvements persist, with 3.4–$3.7\times$ advantages over Full Offload, roughly twofold over COACH and DeltaCNN, and up to $3.6\times$ relative to ROI. Note that the DAVIS dataset used in segmentation is more dynamic than 3DPW in Detect, limiting feature reuse and causing all reuse-based methods in Segment to show less acceleration over Full Offload than in Detect.

Table II
ACCURACY COMPARISON OF DIFFERENT SYSTEMS UNDER DIFFERENT BANDWIDTH CONDITIONS (%).

Bandwidth	Model	ROI	COACH	DeltaCNN	FluxShard
High	Segment	93.0	84.5	97.1	96.8
	Detect	94.2	87.2	99.2	99.0
Medium	Segment	93.0	85.2	97.0	96.3
	Detect	94.2	88.1	99.1	98.8
Low	Segment	93.0	86.0	96.9	95.5
	Detect	94.2	89.0	99.0	98.5

b) Accuracy: Table II further shows the accuracy trade-offs when using different schemes, with the output of the larger version of the model (YOLO11x and YOLO11x-seg [17]) as the ground truth reference. FluxShard maintains accuracy at a high level: for Segment it achieves 95.5–96.8%, corresponding to about 3–4.5% drop, while for Detect it stays between 98.5–99.0%, i.e., within 1–1.5% of the full model. DeltaCNN shows a nearly identical profile, retaining 96.9–97.1% for Segment and 99.0–99.2% for Detect. By contrast, COACH incurs a much larger degradation of around 11–15%, and ROI consistently remains about 6–7% lower than the full model. Overall, FluxShard sustains multi-fold latency gains while bounding accuracy loss within 1–4.5%, a clear improvement over prior baselines.

c) Cache Hit Ratio: Table III presents the cache hit ratios of different methods under varying bandwidth conditions for Segment and Detect. Higher cache hit ratio reflects that more false misses are excluded and more cached features are being reused by each system. COACH rely on whole-frame similarity for early exit, leading to low feature reuse. On dynamic datasets (e.g., DAVIS), their cache hit ratio remains as low as $\sim$1.4% for Segment, but improves to $\sim$25.6% for Detect when dealing with more stable camera scenarios. However, both methods maintain high bandwidth consumption, with COACH requiring up to 10.85 MB/s for Segment in high-bandwidth conditions.

DeltaCNN improves reuse efficiency by applying partial feature caching, achieving hit ratios of $\sim$23% (Segment) and $\sim$73% (Detect). Its bandwidth usage varies significantly across settings, consuming up to 16.57 MB/s in high-bandwidth scenarios but dropping to 4.89 MB/s in low-bandwidth conditions. ROI method only foucuses on the ROI of the frame, which

Table III
CACHE HIT RATIO (%) COMPARISON AND AVERAGE BANDWIDTH CONSUMPTION (MB/S) UNDER DIFFERENT BANDWIDTH CONDITIONS. AVERAGE BANDWIDTH CONSUMPTION IS SHOWN IN PARENTHESES.

Bandwidth	Model	ROI	COACH	DeltaCNN	FluxShard
High	Segment	64.5 (10.05)	1.4 (10.85)	22.4 (6.57)	75.3 (2.05)
	Detect	53.5 (7.42)	25.6 (3.17)	52.8 (4.64)	91.1 (0.26)
Medium	Segment	64.5 (5.58)	1.4 (5.05)	23.1 (3.08)	73.8 (1.04)
	Detect	53.5 (3.38)	25.6 (1.26)	53.4 (2.34)	91.2 (0.24)
Low	Segment	64.5 (0.113)	1.4 (0.67)	23.7 (0.89)	72.5 (0.32)
	Detect	53.5 (0.020)	25.6 (0.092)	53.1 (0.26)	90.7 (0.23)

varies accross different scenes and average to a hit ratio of ∼64.5% for Segment and ∼53.5% for Detect.

FluxShard achieves the highest hit ratio 75.3% for Segment and 91.1% for Detect by leveraging motion-vector-guided cache remapping that maintains high cache locality even under dynamic environments. Importantly, FluxShard maintains the lowest bandwidth consumption across all settings, requiring only 2.05 MB/s at high bandwidth and 0.32 MB/s at low bandwidth while sustaining peak cache efficiency. This demonstrates its ability to adaptively manage feature transmission for efficient bandwidth utilization and improved performance consistency.

The dominant reason for these consistent speedups is that using cache remapping, FluxShard minimizes redundant computation and transmission of costly *false misses* that other methods often treat as cache failures. Instead, only *true misses* are forwarded and fully recomputed, while the majority of aligned regions are served directly from cache. Moreover, activation-guided salience back-projection highlights only the blocks that truly matter for final predictions, ensuring that updates prioritize accuracy-critical regions without wasting bandwidth on irrelevant areas. Together, these mechanisms greatly reduce both compute and communication overhead, explaining why FluxShard sustains several-fold acceleration across network conditions while maintaining high accuracy.

C. Micro-benchmark

a) Cache Hit Ratio under Different scene motions:

b) Computation Time against Cache Hit Ratio:

c) Time Composition of Different Systems: Figure 4 shows the execution time breakdown of ROI, COACH, DeltaCNN, and FluxShard under medium bandwidth conditions for Segment and Detect. FluxShard effectively reduces both transmission and computation time through its motion-vector guided cache remapping and adaptive execution path scheduling. Note that while ROI method uses a tiny yolo11n-detect model to detect the region of interest, it still incurs high overhead since it would need to handle multiple regions of interest when the testing scene is complex. Instead of offloading region of interest like ROI, quantized frames like COACH, or frame deltas like DeltaCNN, FluxShard transmits only motion-triggered regions in a selective manner, significantly reducing transmission overhead. Additionally, by dynamically distributing computation between the edge and the cloud, it minimizes redundant local processing and reduces server-side inference cost. These optimizations allow FluxShard to achieve a lower overall execution time while maintaining high inference accuracy, making it well-suited for real-time video analytics in resource-constrained edge-cloud settings.

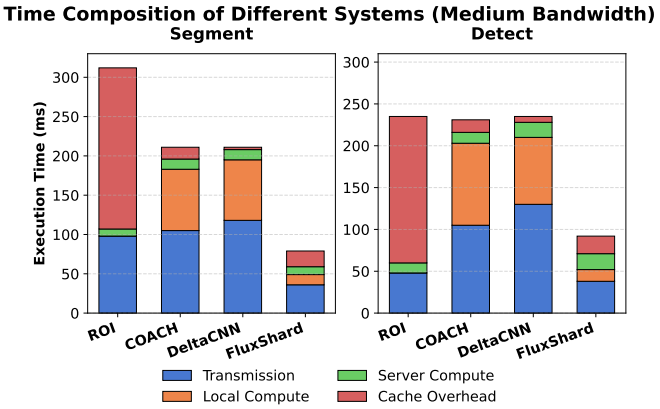

Figure 4. Execution time composition of COACH, ROI, DeltaCNN, and FluxShard under medium bandwidth conditions for Segment and Detect workloads.

Table IV
ABLATION STUDY ON FLUXSHARD'S OPTIMIZATIONS IN MEDIUM BANDWIDTH (SEGMENT). WE TEST THE IMPACT OF REMOVING SALIENCE PROPAGATION (NO-SALIENCE) AND SPATIO SPARSE EXECUTING (DENSE RECOMPUTATION).

Method	E2E Latency (ms)	Cache Hit (%)	Accuracy (%)
Full-Scale FluxShard	94.40	73.8	96.3
No Salience	148.74	53.4	96.5
No-MS-Comp	120.65	73.8	96.3

D. Scalability

We study scalability by varying the number of workers under the *medium* bandwidth setting, focusing on the Segment model as the heavier workload. Medium bandwidth is chosen because high bandwidth masks contention effects, while low bandwidth is already network-bound. Fig. 5 shows that latency increases with more workers due to contention on the shared link, but the growth rate differs sharply across methods. Full Offload grows from 290 ms (1 worker) to 798 ms (4 workers), a 2.8× rise. COACH increases more moderately, 1.9× (228 → 427 ms), while DeltaCNN rises 2.2× (260 → 582 ms). ROI also grows slowly (1.8×, 245 → 434 ms). FluxShard, by contrast, only increases from 94 ms to 159 ms, a 1.7× rise—the flattest curve, and starting from the lowest latency. At 4 workers, FluxShard remains 5.0× faster than Full Offload, 3.7× faster than DeltaCNN, and 2.7× faster than both COACH and ROI. These results confirm that FluxShard sustains both the lowest absolute latency and the slowest relative growth under bandwidth contention.

E. Sensitivity Analysis and Ablation Study

Table IV presents an ablation study under medium bandwidth with the Segment model. Removing salience-guided propagation notably hurts efficiency: end-to-end latency increases from 94 ms to 149 ms while cache hit rate drops from 74% to 53%, showing that accurate salience projection is

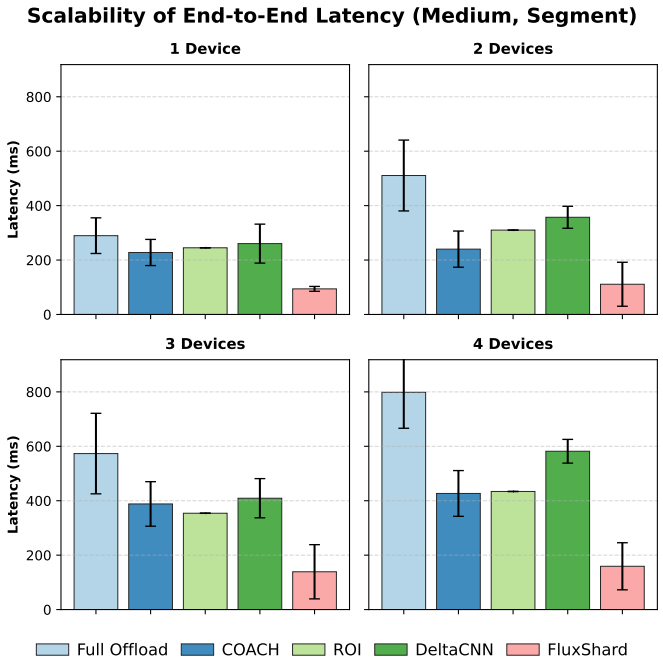

Figure 5. Scalability comparison of end-to-end latency in medium-bandwidth conditions for Segment. FluxShard scales more efficiently, maintaining lower latency growth as devices increase.

essential for reusing the right regions. Disabling the motion-sensitive sparse execution yields a more moderate degradation (121 ms), as recomputation reduces efficiency but leaves hit rate and accuracy unaffected. Overall, both components contribute, with salience propagation being the dominant factor in sustaining high reuse and low latency.

F. Limitations and Future Work

FluxShard is mainly designed around continuous video inputs, where temporal redundancy enables effective reuse. Its efficiency may decrease in cases of rapid scene changes that lower cache hit ratios, and the use of cached features introduces some memory and computation overhead on resource-constrained devices. Future work will consider adaptive cache management and lightweight optimizations to improve robustness and efficiency across a wider range of scenarios.

VIII. Conclusion

We have presented *FluxShard*, a motion-aware video analytics system that reframes feature reuse in dynamic environments as a *motion-induced cache-remapping* problem. By combining motion-vector-guided alignment with salience-driven prioritization and opportunistic freshness, FluxShard minimizes redundant recomputation while sustaining accuracy under camera and object motion. Our edge–cloud implementation demonstrates substantial bandwidth reduction, multi-fold acceleration, and strong scalability across diverse vision tasks, consistently outperforming state-of-the-art baselines. More broadly, FluxShard shows that lightweight motion signals, when paired with task-aware scheduling, provide an effective foundation for efficient and robust video analytics. We believe this perspective opens new opportunities for motion-aware reuse strategies, adaptive caching, and collaborative processing in future resource-constrained, real-time systems.

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] Shahab S Band, Sina Ardabili, Mehdi Sookhak, Anthony Theodore Chronopoulos, Said Elnaffar, Massoud Moslehpour, Mako Csaba, Bernat Torok, Hao-Ting Pai, and Amir Mosavi. When smart cities get smarter via machine learning: An in-depth literature review. *IEEE Access*, 10:60985–61015, 2022.

[3] Sweta Bhattacharya, Siva Rama Krishnan Somayaji, Thippa Reddy Gadekallu, Mamoun Alazab, and Praveen Kumar Reddy Maddikunta. A review on deep learning for future smart cities. *Internet Technology Letters*, 5(1):e187, 2022.

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

[5] Jianguo Chen, Kenli Li, Qingying Deng, Keqin Li, and S Yu Philip. Distributed deep learning model for intelligent video surveillance systems with edge computing. *IEEE Transactions on Industrial Informatics*, 2019.

[6] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[8] John Estrada, Sidike Paheding, Xiaoli Yang, and Quamar Niyaz. Deep-learning-incorporated augmented reality application for engineering lab training. *Applied Sciences*, 12(10):5159, 2022.

[9] 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.

[10] Beining Han, Meenal Parakh, Derek Geng, Jack A. Defay, Gan Luyang, and Jia Deng. FetchBench: A simulation benchmark for robot fetching.

[11] 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.

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2018.

[13] Arash Heidari, Nima Jafari Navimipour, and Mehmet Unal. Applications of ml/dl in the management of smart cities and societies based on new trends in information technologies: A systematic literature review. *Sustainable Cities and Society*, 85:104089, 2022.

[14] Yassine Himeur, Somaya Al-Maadeed, Hamza Kheddar, Noor Al-Maadeed, Khalid Abualsaud, Amr Mohamed, and Tamer Khattab. Video surveillance using deep transfer learning and deep domain adaptation: Towards better generalization. *Engineering Applications of Artificial Intelligence*, 119:105698, 2023.

[15] Junhwa Hur and Stefan Roth. Optical flow estimation in the deep learning age. *Modelling human motion: from human perception to robot design*, pages 119–140, 2020.

[16] Fatemeh Jalali, Morteza Khademi, Abbas Ebrahimi Moghadam, and Hadi Sadoghi Yazdi. Robust scene aware multi-object tracking for surveillance videos. 638:130114.

[17] Glenn Jocher, Jing Qiu, and Ayush Chaurasia. Ultralytics YOLO, January 2023.

[18] Faisal Khan, Saqib Salahuddin, and Hossein Javidnia. Deep learning-based monocular depth estimation methods—a state-of-the-art review. *Sensors*, 20(8):2272, 2020.

[19] Rahima Khanam and Muhammad Hussain. Yolov11: An overview of the key architectural enhancements. *arXiv preprint arXiv:2410.17725*, 2024.

[20] 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.
[21] Zheng Li, Yongcheng Wang, Ning Zhang, Yuxi Zhang, Zhikang Zhao, Dongdong Xu, Guangli Ben, and Yunxiao Gao. Deep learning-based object detection techniques for remote sensing images: A survey. *Remote Sensing*, 14(10):2385, 2022.
[22] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. Edge computing for autonomous driving: Opportunities and challenges. *Proceedings of the IEEE*, 107(8):1697–1716, 2019.
[23] Jinna Lv, Qi Shen, Mingzheng Lv, Yiran Li, Lei Shi, and Peiying Zhang. Deep learning-based semantic segmentation of remote sensing images: a review. *Frontiers in Ecology and Evolution*, 11:1201125, 2023.
[24] 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.
[25] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B. Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–2358, 2017.
[26] Armin Masoumian, Hatem A Rashwan, Julián Cristiano, M Salman Asif, and Domenec Puig. Monocular depth estimation using deep learning: A review. *Sensors*, 22(14):5353, 2022.
[27] Alican Mertan, Damien Jade Duff, and Gozde Unal. Single image depth estimation: An overview. *Digital Signal Processing*, 123:103441, 2022.
[28] Arzoo Miglani and Neeraj Kumar. Deep learning models for traffic flow prediction in autonomous vehicles: A review, solutions, and challenges. *Vehicular Communications*, 20:100184, 2019.
[29] Fábio Eid Morooka, Adalberto Manoel Junior, Tiago FAC Sigahi, Jefferson de Souza Pinto, Izabela Simon Rampasso, and Rosley Anholon. Deep learning and autonomous vehicles: Strategic themes, applications, and research agenda using scimat and content-centric analysis, a systematic review. *Machine Learning and Knowledge Extraction*, 5(3):763–781, 2023.
[30] Khan Muhammad, Amin Ullah, Jaime Lloret, Javier Del Ser, and Victor Hugo C de Albuquerque. Deep learning for safe autonomous driving: Current challenges and future directions. *IEEE Transactions on Intelligent Transportation Systems*, 22(7):4316–4336, 2020.
[31] Debargha Mukherjee, Jim Bankoski, Adrian Grange, Jingning Han, John Koleszar, Paul Wilkins, Yaowu Xu, and Ronald Bultje. The latest open-source video codec vp9 - an overview and preliminary results. In *2013 Picture Coding Symposium (PCS)*, pages 390–393, 2013.
[32] Kien Nguyen, Feng Liu, Clinton Fookes, Sridha Sridharan, Xiaoming Liu, and Arun Ross. Person recognition in aerial surveillance: A decade survey. pages 1–1.
[33] Sankar K Pal, Anima Pramanik, Jhareswar Maiti, and Pabitra Mitra. Deep learning in multi-object detection and tracking: state of the art. *Applied Intelligence*, 51:6400–6429, 2021.
[34] 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.
[35] 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.
[36] Grzegorz Pastuszak and Andrzej Abramowski. Algorithm and architecture design of the h.265/hevc intra encoder. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(1):210–222, 2016.
[37] Roberto Pierdicca, Flavio Tonetto, Marina Paolanti, Marco Mameli, Riccardo Rosati, and Primo Zingaretti. Deepreality: An open source framework to develop ai-based augmented reality applications. *Expert Systems with Applications*, 249:123530, 2024.
[38] 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.
[39] Pablo Fernández Pérez, Claudio Fiandrino, and Joerg Widmer. Characterizing and modeling mobile networks user traffic at millisecond level. In *Proceedings of the 17th ACM Workshop on Wireless Network Testbeds, Experimental evaluation & Characterization*, WiNTECH '23, pages 64–71. Association for Computing Machinery.
[40] Imran Qureshi, Junhua Yan, Qaisar Abbas, Kashif Shaheed, Awais Bin Riaz, Abdul Wahid, Muhammad Waseem Jan Khan, and Piotr Szczuko. Medical image segmentation using deep semantic-based methods: A review of techniques, applications and emerging trends. *Information Fusion*, 90:316–352, 2023.
[41] Niri Rania, Hassan Douzi, Lucas Yves, and Treuillet Sylvie. Semantic segmentation of diabetic foot ulcer images: dealing with small dataset in dl approaches. In *Image and Signal Processing: 9th International Conference, ICISP 2020, Marrakesh, Morocco, June 4–6, 2020, Proceedings 9*, pages 162–169. Springer, 2020.
[42] Yulan Ren, Yao Yang, Jiani Chen, Ying Zhou, Jiamei Li, Rui Xia, Yuan Yang, Qiao Wang, and Xi Su. A scoping review of deep learning in cancer nursing combined with augmented reality: The era of intelligent nursing is coming. *Asia-Pacific Journal of Oncology Nursing*, 9(12):100135, 2022.
[43] Zhe Ren, Junchi Yan, Bingbing Ni, Bin Liu, Xiaokang Yang, and Hongyuan Zha. Unsupervised deep learning for optical flow estimation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.
[44] Stefano Savian, Mehdi Elahi, and Tammam Tillo. Optical flow estimation with deep learning, a survey on recent advances. *Deep biometrics*, pages 257–287, 2020.
[45] GSDMA Sreenu and Saleem Durai. Intelligent video surveillance: a review through deep learning techniques for crowd analysis. *Journal of Big Data*, 6(1):1–27, 2019.
[46] Xudong Sun, Pengcheng Wu, and Steven CH Hoi. Face detection using deep learning: An improved faster rcnn approach. *Neurocomputing*, 299:42–50, 2018.
[47] Aditya Tandon, Rajveer Shastri, Mantripragada Yaswanth Bhanu Murthy, Parismita Sarma, P.N. Renjith, and Masina Venkata Rajesh. Video streaming in ultra high definition (4k and 8k) on a portable device employing a versatile video coding standard. *Optik*, 271:170164, 2022.
[48] 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.
[49] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003.
[50] Qi Wu, Zipeng Fu, Xuxin Cheng, Xiaolong Wang, and Chelsea Finn. Helpful DoggyBot: Open-world object fetching using legged robots and vision-language models.
[51] Xiongwei Wu, Doyen Sahoo, and Steven CH Hoi. Recent advances in deep learning for object detection. *Neurocomputing*, 396:39–64, 2020.
[52] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in neural information processing systems*, 34:12077–12090, 2021.
[53] Jun Zhang, Mina Henein, Robert Mahony, and Viorela Ila. VDO-SLAM: A visual dynamic object-aware SLAM system.
[54] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020.

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.