Juice: Lightweight Foreground Prediction for On-Camera Surveillance Video Compression

Jiajun Yan[1], Hongzi Zhu[1*], Shan Chang[2], Li Li[1], Minyi Guo[1]
[1]Shanghai Jiao Tong University, [2]Donghua University
{yanjiajun, hongzi}@sjtu.edu.cn

***Abstract*—Video surveillance plays a crucial role in modern society, fostering safer, smarter, and more efficient environments. However, it is of great challenge to transmit, store and analyze city-scale surveillance videos due to the massive amounts of data. In this work, we propose *Juice*, a lightweight surveillance video compression scheme that can be implemented on H.265-compliant cameras. The core idea of Juice is to effectively utilize the CU (Coding Unit) division information generated during the encoding process to predict tiles with foreground objects in each frame. Furthermore, redundant background tiles between frames are removed to minimize the compressed video size without compromising the downstream surveillance detection accuracy. We collect a real-world transportation traffic surveillance video datasets, consisting of 541 video clips recorded at 42 distinct locations in different light and weather conditions. We implement Juice and conduct extensive trace-driven simulations. The results demonstrate that Juice is lightweight and can process at least 32 FPS at a resolution of 1920×1080 on a single-core common CPU and can achieve an average 78.2% compression rate. Furthermore, Juice has little impact on downstream AI detection algorithms. Specifically, the object detection on compressed videos undergoes a minor 1% accuracy drop, compared to detection on uncompressed videos.**

***Index Terms*—surveillance video compression, H.265 encoding, on-camera, foreground prediction, differential compression**

I. Introduction

By monitoring public spaces, businesses, and critical infrastructure, video surveillance systems, powered by artificial intelligence (AI) algorithms, have been extensively used across various domains of daily life and industrial operations in modern society, ranging from traffic monitoring [1]–[3] and crowd control [4] to behavior analysis [5] and crisis management. However, the massive scale of video data generated by video surveillance systems poses significant challenges in terms of data transmission and storage. For example, the public video surveillance system in a medium-sized city contains tens of thousands of cameras, generating tremendous data volume of over 100 TB daily. It is the key to effectively compress surveillance videos on cameras with light computation cost without affecting the performance of downstream AI-based detection algorithms.

In the literature, existing surveillance video compression methods fall into two categories, *i.e.*, region-of-interest (ROI)-based and reconstruction-based. ROI-based compression methods require DNN model inference to identify ROIs in each video frame. They use frame masking [6] [7], mixed-resolution [8] and background reusing [9] to compress video, or only reserve some of the frames [10] [11]. Though these methods may preserve the detection accuracy of downstream AI algorithms, exciting model inference on cameras requires prohibitive computational cost [12], [13]. In contrast, reconstruction-based compression methods transmit low-quality videos to a server, where super-resolution reconstruction is performed to recover high-resolution videos [14] or object detection is first conducted and then regions that cannot be recognized are required to be retransmitted in high resolution [15]. These methods rely on the powerful computing capacity on the server but cannot guarantee the performance of downstream AI detection algorithms [16]. As a result, there is no successful surveillance video compression scheme, to the best of our knowledge, that is lightweight and can be deployed on cameras while having no impairment to downstream tasks.

In this work, we propose a simple yet effective surveillance video compression scheme, called *Juice*, which can be implemented on H.265 compatible cameras. We have two key observations about the unique characteristics of H.265 encoding. First, video frames can be divided into multiple small rectangular regions, referred to as *tiles*, which can be independently encoded and decoded. Second, a new hierarchical CTU (Coding Tree Unit) structure is employed in encoding a tile, which assigns denser CUs (Coding Unit) at regions with richer texture. The core idea of Juice is to identify tiles with objects of interest (referred to as foreground tiles) solely based on the CU division variations, without the need of executing deep neural network (DNN) model inference. As a result, foreground tiles are normally encoded and reserved while redundant tiles without any object of interest (referred to as background tiles) are removed, significantly reducing the volume of the resulting video.

The Juice design faces two main challenges. First, CU division patterns exhibit ambiguity, making it hard to distinguish target regions from those noisy regions with rich texture. To tackle this challenge, we observe that objects of interest are intermittently mobile whereas textural background objects are static in most surveillance scenarios. Therefore, we consider both spatial and temporal characteristics of CU division. Specifically, we divide video frames into small blocks of fixed size and score the CU division density in the unit of block. By analyzing the variance of density score of each block and the correlation of density score between neighboring blocks, target objects can be accurately recognized.

Hongzi Zhu is the corresponding author of this paper.

Second, it is non-trivial to maintain high-quality of the compressed video in terms of both human and machine visions. Though background tiles contains no mobile objects, important static objects may exits, *e.g.*, a stop vehicle or a traffic sign. To deal with this challenge, we devise a tile removal and recover mechanism. Specifically, in H.265, each tile in each GOPs (Group Of Pictures) is independently encoded. In Juice, each tile in each GOPs is labelled as a foreground or background tile. Given a background tile in the current GOP, if the corresponding tile in the previous GOP is labelled as a foreground tile, this background tile is marked as a new background tile and will be reserved in the compressed video; otherwise, this background tile is considered redundant and will be removed from the video. In this way, when reconstructing a compressed video, new background tiles can be reused and filled into previously removed locations.

We implement Juice based on the open-source H.265 encoder Kvazaar [17] on a Ubuntu 18.04 desktop with an Intel i7-7800X CPU. Juice is lightweight and can process at least 32 FPS at a resolution of 1920×1080 on a single-core common CPU. We collect a real-world transportation traffic surveillance video datasets, consisting of 541 video clips recorded at 42 distinct locations in different light and weather conditions. Through extensive trace-driven simulations, the results demonstrate that Juice outperforms the state-of-the-art model-inference based compression method MVC [18]. Specifically, the average compression rate of Juice and MVC is 78.2% and 70.3%, respectively. Moreover, compared to object detection results on uncompressed videos, the average detection precision and recall on compressed videos generated using Juice are 86.9% and 79.5%, both of which undergoes a minor accuracy drop of less than 1%. In contrast, the detection precision and recall on compressed videos generated using MVC undergoes a 8.3% and 3.7% drop, respectively.

We highlight our main contributions made in this work as follows:

- A lightweight foreground prediction scheme utilizing the H.265 CU division information is proposed, which suits the deployment on cameras.
- An effective surveillance video compression and reconstruction mechanism is devised, friendly to both human and machine visions.
- An extensive surveillance video dataset is collected and annotated for public study and extensive evaluation is conducted, the results of which demonstrate the efficacy of Juice.

II. Preliminaries & Problem Definition

A. H.265 Video Coding Scheme

H.265, also known as HEVC (High Efficiency Video Coding), represents the next-generation video coding standard developed jointly by ITU-T and ISO/IEC [19] as the successor to H.264/AVC. As illustrated in Figure 1, a video clip can be divided into multiple GOPs (Group Of Pictures). Each GOP consists of one I-frame (intra-coded frame), a

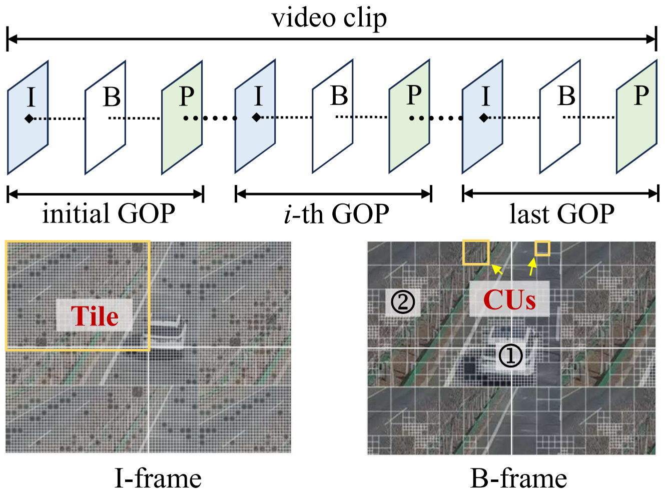


Fig. 1: Introduction to H.265 encoding scheme, where each frame can be divided into independently encoded tiles and each tile can be further organized into hierarchical quad-tree structures of CUs, gaining a trade-off between high visual quality and minimal video bitrate.

set of interweaving B-frames (bi-directional predictive-coded frames) and P-frames (predictive-coded frame), respectively. Specifically, I-frames are encoded solely using intra-frame prediction without relying on information from other frames. I-frames serve as key frames, effectively preventing error propagation throughout the video clip. P-frames are a type of inter-coded frames, which store only the differences (motion-compensated residuals) from a previous reference frame (typically an I-frame or another P-frame). B-frames implement a more sophisticated bi-directional prediction mechanism that achieves higher compression by referencing both past and future frames.

In contrast to H.264's slice-based partitioning, H.265 introduces a *tile*-based division mechanism, where each frame is segmented into independent rectangular regions, referred to as tiles. The same tile throughout all frames in one GOP can be independently encoded and decoded, making H.265 particularly suitable for real-time encoding of high-resolution video on multi-core CPUs or hardware-accelerated platforms. In addition, a tile contains an integer number of CTU (Coding Tree Unit), which can be recursively divided into different-sized CUs (Coding Units) in a quad-tree structure, ranging from 64×64 pixels to 8×8 pixels [20]. The CU division follows a RDO (Rate-Distortion Optimization) process, where the encoder evaluates different partitioning configurations and selects the one that minimizes the rate-distortion cost [21]. This allows the encoder to adaptively select the optimal division strategy based on local video content characteristics.

For instance in Figure 1, each frame is segmented into four tiles and each tile are further divided into CUs of different sizes. As illustrated in the B-frame, finer division is utilized in textured regions, *i.e.*, ① an emerging vehicle in the foreground and ② static trees in the background, to preserve visual quality while larger CUs are employed in homogeneous regions to reduce bitrate. In addition, it can also be seen that I-frames have significantly denser CU division for its intra-coded nature.

A fundamental architectural improvement in H.265 is the replacement of H.264's fixed macroblock structure with a more flexible CTU division mechanism. The CTU can be recursively divided into different-sized CU in a quadtree structure, ranging from 64x64 pixels to 8x8 pixels [20]. This allows the encoder to adaptively select the optimal division strategy based on local video content characteristics. Each CU can be further subdivided into PUs (Prediction Units) and TUs (Transform Units) [22] for motion prediction and transform coding [23]. The CU division process is guided by RDO (Rate-Distortion Optimization), where the encoder evaluates different partitioning configurations and selects the one that minimizes the rate-distortion cost [21]. This adaptive approach enables HEVC to efficiently handle varying content complexity: employing larger CU in homogeneous regions to reduce bitrate while utilizing finer division in textured areas to preserve visual quality.

In summary, if tiles with target objects of interest can be identified, an effective surveillance video compression algorithm can designate a small compression ratio to such tiles with other tiles heavily compressed or even discarded.

B. System Model and Problem Definition

We consider the following two types of entities in the system:

- **Surveillance cameras**: Cameras encode raw video streams in line with the tile-based H.265 standard and the CU division information of each frame is available. In addition, cameras have limited computing power and can only conduct basic arithmetic and logic operations. Moreover, cameras have stable power supply and connect to a data center via dedicated wired or wireless channels.
- **Data center servers**: Data center servers receive, store and decode these encoded surveillance videos. Moreover, they have sufficient computing power to conduct various surveillance detection tasks using AI algorithms.

The surveillance video compression problem is defined as: how to *minimize the surveillance video size on cameras* for efficient data transmission and storage, while *guaranteeing the rigid accuracy requirement of downstream AI surveillance detection tasks* executed at data center servers.

III. Design of Juice

A. Overview

Given the key observation that the H.265 video encoder tends to employ dense CU division for regions with rich texture, the core idea of Juice is to predict tiles with objects of interest solely based on the CU division information and remove those redundant background tiles. Moreover, compressed videos can be well recovered at a server and have minor impact on both human and machine visions. As shown in Figure 2, the architecture of Juice consists of the following three main components, *i.e.*, *Predicting the Foreground* and *Removing Redundant Tiles* at a camera and *Reconstructing Compressed Video* at a data center server.

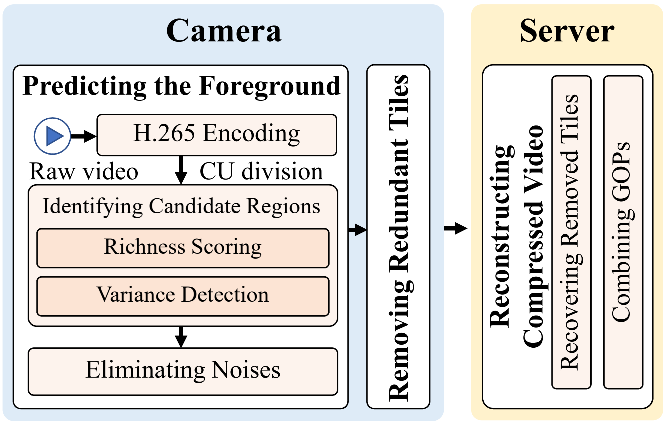


Fig. 2: The architecture of Juice, where foreground regions can be predicted based on CU division information and used to differentially compress the video on cameras.

Predicting the Foreground (PTF). PTF utilizes the CU division information to identify regions with dense CUs and eliminates those noise regions caused by static background with rich texture. As a result, tiles with the predicted foreground in each GOP are identified.

Removing Redundant Tiles (RRT). RRT reserves all foreground tiles and those new background tiles among GOPs, and remove those redundant background tiles, reducing the ultimate compressed video size.

Reconstructing Compressed Video (RCV). RCV recovers a compressed video by reusing previously reserved background tiles when a background tile in the current GOP is missing. Then, all recovered GOPs are merged into one complete video clip.

B. Predicting the Foreground

As shown in Figure 1, due to the intra-coded nature of I-frames and the forward-only reference of P-frames, their CU partitioning appears significantly denser compared to B-frames, with reduced discriminative capability for moving objects. Moreover, the number of B-frames are dominant. Therefore, Juice chooses B-frames to study. After H.265 encoding of raw camera footage, each frame has CU division scheme, where the size of a CU ranges hierarchically from 64×64 down to 8×8 pixels. The PTF component analyzes the CU division information to predict foreground objects by first identifying regions with dense CUs and distinguishing true foreground objects and background noises.

1) Identifying Candidate Regions: To identify regions with rich texture, we divide each frame into blocks of 16×16 pixels and assign a CU density score to each block. Specifically, we denote $\mathcal{S}_t^i$ as the density score of a block B_i at the t-th frame, defined as

$$\mathcal{S}_t^i = \begin{cases} 0.5, & w_i = 16 \times 16 \\ 0.25, & w_i = 32 \times 32 \\ 0.05, & w_i = 64 \times 64 \end{cases} \tag{1}$$

where w_i denotes the size of the corresponding CU in this frame where block B_i belongs.

As an example shown in Figure 3, the density scores of five example blocks (B_1-B_5) are depicted when a vehicle is traversing a road segment. Specifically, before timestamp t_1, the vehicle has not arrived at block B_1 and B_2 yet, and $\mathcal{S}_{t_1}^1$ and $\mathcal{S}_{t_1}^2$ (depicted as dark squares in Figure 3) remain low. After the vehicle appears in both B_1 and B_2, for instance at timestamp t_2, $\mathcal{S}_{t_2}^1$ and $\mathcal{S}_{t_2}^2$ rise up until the vehicle leaves both blocks. The reason is that the encoder tries to capture the texture change by employing and revoking denser CUs at B_1 and B_2 when the vehicle emerges and leaves both blocks, respectively. It can also be seen that $\mathcal{S}^3$ has quite similar patten with $\mathcal{S}^2$ except there is an obvious lag due to the late arrival of the vehicle at B_3. In addition, it can be seen that when the vehicle traverses B_1 and B_2, $\mathcal{S}^4$ is affected because CU is recursively partitioned using a quad-tree structure which may affect neighboring blocks with no texture change.

To effectively detect density changes of a block, we devise a *windowed variance* of density score for a block B_i across a sliding window of $2N+1$ frames centered at frame t, defined as

$${\sigma^2}_t^i = \frac{1}{2N+1} \sum_{k=t-N}^{t+N} \left(\mathcal{S}_k^i - \mu_t^i,\right)^2 \tag{2}$$

where μ_t^i denotes the mean density score of this block over this sliding window. A threshold α can be configured to determine a significant density changes of a block. For instance when $\alpha = 0.04$ as shown in Figure 3, the texture changes in B_1, B_2 and B_3 caused by the moving vehicle can be effectively identified (shown as blue triangles) ahead of the arrival of the vehicle and retained during the whole process. This provides extra prevention from missing objects of interest. In addition, affected blocks (*e.g.*, B_4) usually present low windowed variance and can be filtered out.

2) Eliminating Noises: From Figure 3, it is also interesting to see that $\mathcal{S}^5$ varies over time, which is caused by random texture variations when dealing with static textural objects (*e.g.*, a tree in this example) in the background. Moreover, the windowed variance of B_5 can exceed the threshold α in frames 13-15, confusing the foreground detection.

To deal with noisy background blocks, we leverage the impact of the *spatio-temporal locality* of object movement to the CU division scheme, which demonstrates similar density score patterns among blocks (*e.g.*, B_1 and B_2) experiencing different parts of a moving object at the same time and among blocks (*e.g.*, B_2 and B_3) experiencing the same part of a moving object in temporal order. We calculate *extended-window Pearson correlation* to depict this similarity. The

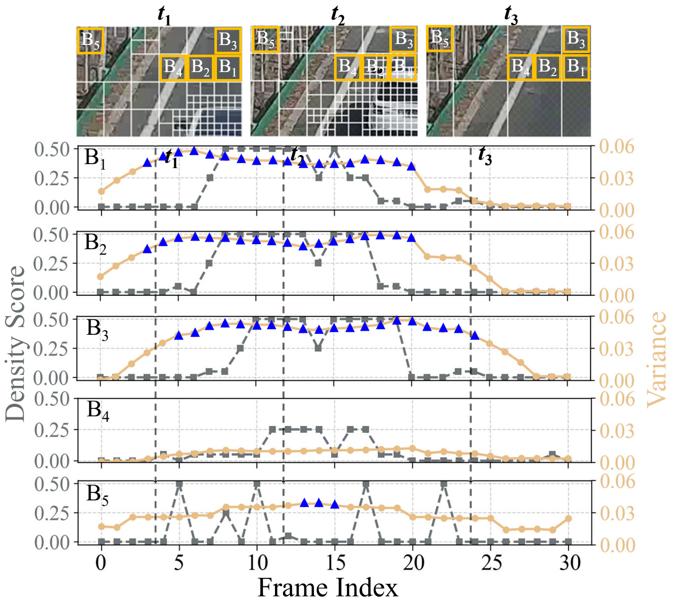


Fig. 3: An illustration of how density scores of five example blocks change when a vehicle is traversing a road segment.

Pearson correlation between two continuous variables X and Y is defined as

$$\begin{aligned} P(X, Y) &= \frac{\mathrm{Cov}(X, Y)}{\sigma_X \sigma_Y} \\ &= \frac{\mathbb{E}[(X - \mu_X)(Y - \mu_Y)]}{\sqrt{\mathbb{E}[(X - \mu_X)^2]}\sqrt{\mathbb{E}[(Y - \mu_Y)^2]}} \end{aligned} \tag{3}$$

For a candidate block B_i whose windowed variance exceeds threshold α from frame m to frame n, we set $[m-T, n+T]$ as the sliding window to mitigate the impact of lag. For the neighborhood B_j of B_i, the *extended-window Pearson correlation* $\rho_{ij}(m,n)$ between B_i and B_j is defined as

$$\rho_{ij}(m,n) = \max_{t \in [-T,T]} P\left(\mathcal{S}^i[m:n], \mathcal{S}^j[m-t:n+t]\right) \tag{4}$$

The block B_i is classified as a genuine foreground block when it has at least one adjacent block with *extended-window Pearson correlation* exceeding the predefined threshold β. The configuration of β will be examined in Subsection IV-B. The process of calculating correlation is time-consuming. However, we only compute the correlation between each candidate block and its 3×3 neighborhood, which incurs acceptable computational overhead.

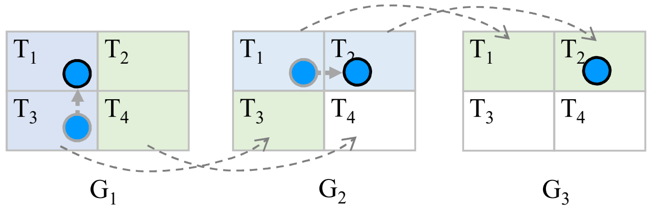


Fig. 4: Tile removal and recording process.

C. Removing Redundant Tiles

After predicting the foreground, each tile is classified as either a foreground tile or background tile. The RRT component reserves foreground tiles and only newly emerged background tiles across GOPs while eliminating redundant background tiles, thereby achieving significant video compression.

As an example shown in Figure 4, an moving object depicted as a blue dot first traverses from T_3 to T_1 during the first GOP G_1, progresses from T_1 to T_2 during the second GOP G_2, and ultimately comes to a stop at T_2 during the third GOP G_3. As a result, in G_1, T_1 and T_3 are labelled as foreground tiles (depicted as blue tiles) and T_2 and T_4 are labelled as background tiles. As T_2 and T_4 are within the first GOP, they are considered as new background tiles (depicted as green tiles). Similarly, in G_2, T_1 and T_2 are foreground and T_3 and T_4 are background. As T_3 in G_1 is foreground, T_3 in G_2 is considered a *new* background. In contrast, T_4 in G_2 is considered redundant because T_4 in G_1 is already a background tile, which should be removed from the compressed video. Similarly, both T_1 and T_2 in G_3 are considered new whereas T_3 and T_4 in G_3 are considered redundant.

Juice utilizes MP4 file tracks for both tile removal and recovery. Specifically, we employ the open-source tool GPAC [24] to split each encoded video tile into separate tracks within the MP4 container. This design allows efficient tile removal and recovery through simple track deletion or addition in the MP4 file.

D. Reconstructing Compressed Videos

After RRT processing, the compressed video is transmitted to the server and stored as GOPs with partially missing tiles. The RCV component reconstructs the original video by reusing background tiles and merging GOPs, ensuring both optimal performance for downstream tasks and enhanced viewing experience for human observers.

1) Recovering Removed Tiles: The system reuses background tiles reserved from preceding GOPs, sequentially recovering missing tiles by referencing corresponding tracks from prior GOPs in chronological order. This process ensures pixel-perfect reconstruction while maintaining spatial consistency with the source footage.

2) Combing GOPs: For broad compatibility with most video players, we subsequently combine multiple tracks into a unified stream in each MP4 file. Finally, the processed GOPs are merged into a seamless, playable video file using ffmpeg [25], completing the restoration pipeline while maintaining full visual fidelity.

IV. Performance Evaluation

A. Methodology

1) Implementation: We implement both camera and server components of Juice based on the open-source H.265 encoder Kvazaar [17] on a Ubuntu 18.04 desktop with an Intel i7-7800X CPU. Particularly, to mimic the computing capability of a camera, such as HiSilicon's Hi3519A processor [26], the camera components run using only one core of the CPU. We set the windowed variance threshold α to an empirical value of 0.04.

2) Dataset: In collaboration with the government sector, we construct a transportation traffic surveillance video dataset. Specifically, 541 H.265 encoded video clips of one minute with a resolution of 1920×1080 from 42 traffic monitoring points across Daqing, a northeast city in China, are collected from February 7 to June 10, 2025. The GOP of all clips adopts the default 17-frame configuration which includes one I-frame, 15 B-frames and one P-frame. We employ annotators to label all objects in all video clips as ground truth for evaluating the performance of downstream object detection task. The dataset captures diverse road types (one-way highways, two-way streets, and complex urban intersections) under varying conditions, including different weather patterns (clear, rainy, and snowy conditions), illumination conditions (daytime, dusk, and night) and traffic densities.

3) Candidate Methods: We compare Juice with the following candidate methods:

- **MVC:** MVC (Model Inference-based Video Compression) leverages a deep neural network for foreground-background segmentation of video content. The system processes the detected foreground regions using standard encoding techniques while applying aggressive compression to background areas through deep learning-based quality reduction [9] [18].
- **UVC:** UVC (Unified Video Compression) employs increased QP (Quantization Parameter) during encoding to reduce video quality uniformly, thereby achieving compression.
- **NVC:** NVC (None Video Compression) does not use any additional compression methods. It only encodes and serves as a comparison for the accuracy of downstream tasks.

4) Performance Metrics: We conduct downstream vehicle detection task using YOLOv11 [27]. The detection results are compared against manually annotated ground truth to obtain precision and recall. We consider the following metrics,

- **Precision**: Precision measures the proportion of downstream detection task correctly identified positive instances among all instances predicted as positive. A high precision means a substantial amount of compression.
- **Recall**: Recall measures the proportion of downstream detection task correctly identified positive instances among all actual positive instances. A high recall means that little information is lost.
- **Compression ratio**: Compression ratio measures the reduction in data size, expressed as the ratio of the compressed size to the original size.
- **SSIM**: SSIM (Structural Similarity Index Measure) measures the quality of digital images and videos by considering changes in structural information, luminance, and contrast. Higher values indicate better fidelity.

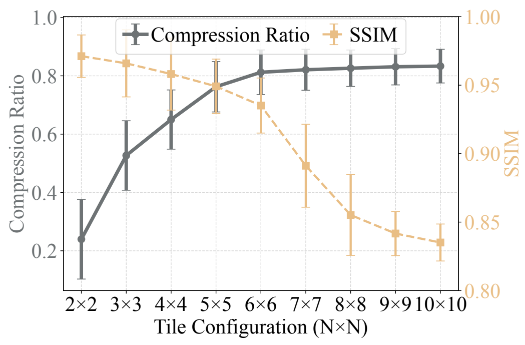


Fig. 5: Video size and SSIM with tiling configuration.

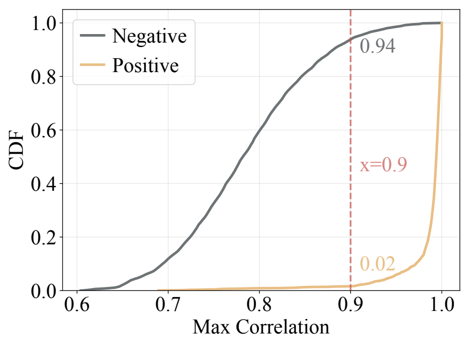


Fig. 6: Correlation of positive and negative samples.

B. Parameter Configuration

We first investigate the parameter configurations for both tiling patterns and correlation thresholds, conducting experiments on 20 representative video clips selected from our constructed dataset.

1) Tiling configuration: We encode videos of varying scene complexities using different tiling configuration ranging from 2×2 to 10×10 grids. Each configuration was systematically evaluated through our complete processing pipeline, with record for both compression ratio and reconstruction quality (SSIM). As illustrated in the Figure 5, our experimental results demonstrate that finer tile partitioning leads to progressive reduction in SSIM values while simultaneously improving compression ratio, which aligns with our theoretical expectations. As the tile division becomes more dense, Juice can more accurately reserve the foreground regions, thereby enhancing compression ratio. However, the independent encoding of each tile introduces a counteracting effect: the compression ratio within individual tiles decreases as their spatial dimensions shrink, resulting in diminishing returns for overall size reduction. Furthermore, excessive tile replacement was observed to cause noticeable visual artifacts that degrade perceptual quality. After careful consideration between compression gain and quality preservation, we chose a 6×6 tile configuration in the subsequent experiments.

2) Correlation Analysis Threshold: Figure 6 presents the correlation between blocks and their neighbor. We classify blocks into positive and negative categories based on the presence or absence of moving objects. The result reveals that as the correlation threshold increases, the filtering of noise becomes stronger, but at the same time, it also mistakenly filters out real objects. This enables threshold β adjustment to optimally balance compression efficiency and information preservation across different operational scenarios. It is noteworthy that since moving objects typically span multiple blocks in video sequences, the actual proportion of affected objects is substantially lower than this block-level error rate. For all subsequent experiments in this study, we selected $\beta = 0.9$ as the correlation threshold.

C. Overall Performance

In this experiment, we compare the performance of candidate methods across four metrics on the entire dataset. In practical applications, the detection accuracy for nearby vehicles carries significantly greater importance and value than for distant vehicles. Therefore, we divide the detection results into near and far based on the position in the video. For Juice and MVC, we use the default QP setting of 22 in kvazaar encoder. For UVC, we adjust the QP to ensure that each encoded video has a size similar to that of Juice-compressed videos.

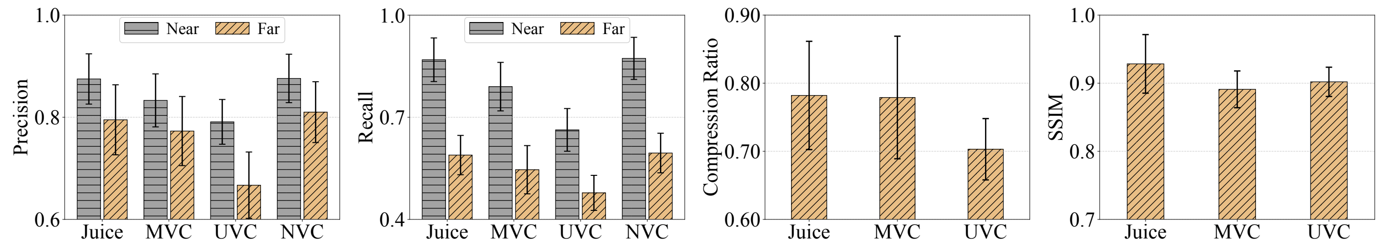
The experimental results are shown in Figure 7. At the same compression ratio, UVC perform significantly worse than Juice in downstream tasks, with 8.4% lower near-range precision, 20.6% lower near-ranger recall, and a 3% higher SSIM. This demonstrates that while traditional methods can achieve video compression, they do so at the cost of severe quality degradation. Therefore, we exclude this method from subsequent comparative experiments under different scenarios. Compared to MVC, Juice improves near-range precision by 4.1% and near-range recall by 7%, while maintaining a difference of less than 0.4% compared to the original video. Additionally, its far-range performance also differs by less than 1.2% from the original. With almost no drop in downstream task performance, Juice achieves a 78.2% video compression rate, 7.9% higher than UVC.

D. Comparison under Different Scenarios

In order to verify the performance of the system in different scenarios, we categorize the dataset according to two environmental parameters: illumination conditions and traffic density levels.

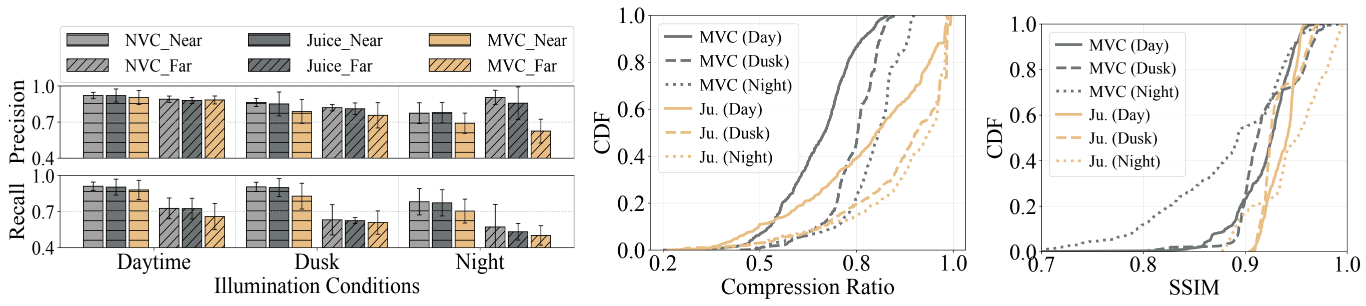
1) Illumination Conditions: In this experiment, we classify our dataset into three illumination scenarios : daytime (well-lit), dusk (variable shadows), and nighttime (low-light). We investigate the impact of illumination conditions on the performance of Juice and MVC.

Figure 8 shows the four metrics of Juice and MVC on three illumination scenarios. Juice consistently outperforms MVC across all lighting conditions and evaluation metrics. Juice maintains nearly identical precision and recall to original footage in close-range scenarios, with only minor degradation



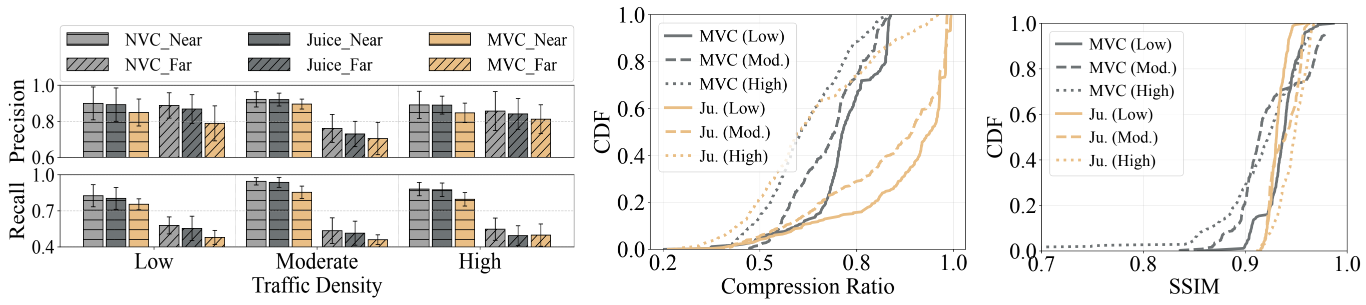
(a) Precision (b) Recall (c) Compression ratio (d) SSIM

Fig. 7: Overall performance.



(a) Precision and Recall (b) Compression Ratio (c) SSIM

Fig. 8: Performance comparison of our method and the baseline under different illumination conditions.



(a) Precision and Recall (b) Compression Ratio (c) SSIM

Fig. 9: Performance comparison of our method and the baseline under different traffic density levels.

in distant nighttime areas. This confirms our method's effectiveness in preserving critical visual information under varying illumination without compromising detection accuracy. Notably, nighttime reconstruction achieves superior quality (0.945 SSIM), showing 6.4% improvement over baseline. This stems from minimal artifacts when replacing tiles in static low-light environments. Daytime (0.936) and dusk (0.931) scenarios present greater reconstruction challenges due to dynamic lighting and shadows, yet Juice still surpasses comparison methods in all conditions.

2) Traffic Density Level: Video compression performance is significantly influenced by traffic density, with higher vehicle concentrations reducing the number of removable tiles and consequently decreasing compression efficiency. To systematically evaluate this effect, we categorize the dataset into three traffic density levels (high, medium, and low) based on instantaneous vehicle counts in surveillance. We investigate the impact of traffic density level on the performance of Juice and MVC.

Figure 9 show consistent downstream task performance across all density levels compared to the NVC, with negligible variations in SSIM. However, significant differences emerge in compression ratio: Juice achieves compression rates of 87.2% (15.8% better than baseline) under low density, 82.6% (14.9% superior) for moderate density, and 63.6% (marginally 1.4% better) in high-density scenarios. These findings reveal two key insights. First, traffic density substantially impacts Juice's compression efficiency, as expected given the funda-

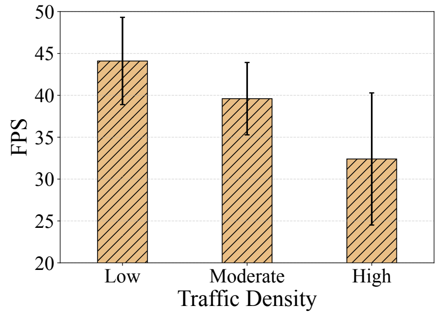


Fig. 10: FPS achieved in different traffic density.

mental relationship between moving objects and our tile-based compression approach. Second, even under maximum traffic density, Juice maintains a remarkable 63.6% compression rate, demonstrating its robustness and superior performance under challenging, real-world traffic conditions.

E. Efficiency Analysis

To satisfy real-time on-camera compression demands, we evaluate the computational efficiency of Juice. As processing time scales with potential moving objects, we measure FPS (Frames Per Second) of Juice across traffic densities. Figure 10 shows that the FPS of Juice is 44.1 in low , 39.6 in Moderate and 32.4 in high density. This is in line with expectations. The increase in traffic density leads to an increase in the proportion of foreground tiles and the tiles that need to be retained for restoration. Nevertheless, the juice maintains over 30 fps in all scenarios, comfortably meeting standard traffic surveillance requirements (25-30 fps).

V. Discussion

A. Implement Juice on Cameras

Juice's algorithm has low theoretical computational needs and relies only on CU partitioning data from H.265 encoding. However, practical deployment encounters a major hardware barrier: current surveillance cameras use integrated hardware encoders that lack interfaces for CU data extraction. This prevents direct hardware-level implementation. A software-based workaround is possible but offers no practical advantage over server-side software encoding. The issue could be resolved if future chip designs exposed CU information interfaces.

B. Information Loss for Distant Objects

During motion detection, distant vehicles occupying few pixels may be lost as their tiles are removed. This is acceptable as such small targets are typically undetectable in downstream tasks, and continuous vehicle trajectories ensure sufficient data from closer ranges. Traffic lights pose a specific challenge, as instantaneous state changes might be missed, leading to incorrect colors in reconstructed video. A future solution could be to predefine and preserve fixed regions around traffic signals.

VI. Related Work

Existing work on surveillance video compression falls into two categories, ROI-based and reconstruction-based.

ROI-based compression: AsaMask [6] enables machine-centric video streaming by dynamically masking frames to transmit only object-relevant regions. MRIM [8] employs saliency-aware non-uniform downscaling. Reducto [10] performs on-camera frame filtering by adjusting feature thresholds based on accuracy needs. Clownfish [11] introduces edge-cloud symbiosis, using a lightweight edge model for real-time processing and offloading key frames to the cloud. UVCNet [9] proposes unsupervised video compression by dynamically separating foreground and background online for efficient encoding. The above methods require full video decoding. TileClipper [28] selectively transmits frame regions (tiles) based on bitrate correlation. CRUCIO [18] employs reshaped asymmetric autoencoders to filter and compress frames spatially and temporally. However, they require model inference on the server or client side.

Reconstruction-based compression: CASVA [29] employs a reinforcement learning mechanism to adaptively adjust video encoding parameters based on varying bandwidth conditions. AccMPEG [30] optimizes video streaming for edge-to-server DNN inference by predicting accuracy gradients at the macroblock level using a lightweight model, enabling dynamic encoding. These two methods are both aimed at maintaining the accuracy of the inference, it requires them to rely on the server side to get the right parameters. DDS [15] introduces a server-driven video streaming approach where the server-side DNN analyzes a low-quality video stream and selectively requests high-quality updates for critical regions, optimizing bandwidth efficiency while maintaining or improving inference accuracy. Similarly, CloudSeg [14] introduces an edge-to-cloud vision analytics framework that transmits low-resolution video streams and employs task-aware super-resolution on the cloud side, achieving bandwidth reduction with minimal accuracy loss. These two methods require a significant amount of computing resources for the server-side reconstruction.

VII. Conclusion

In this paper, we have proposed Juice, a simple yet effective surveillance video compression scheme. Juice is lightweight and has no requirement for high computation power. Furthermore, Juice is H.265 compatible making it easy to integrate into future camera DSP SoC. Juice can significantly reduce 78.2% the surveillance video volume on average. It has little impact to downstream AI detection algorithms and achieves less than 1% accuracy drop in terms of both precision and recall for the object detection task, compared to using uncompressed video.

Acknowledgment

This work was supported in part by the Natural Science Foundation of China (Grants No. 62432008, 62472083), the AI-Enhanced Research Program of Shanghai Municipal Education Commission (Grant No. SMEC- AI-DHUZ-01).

References

[1] O. Elharrouss, N. Almaadeed, and S. Al-Maadeed, "A review of video surveillance systems," *Journal of Visual Communication and Image Representation*, vol. 77, p. 103116, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1047320321000729

[2] Y. Li, H. Zhu, Z. Deng, Y. Cheng, Z. Zheng, L. Zhang, S. Chang, and M. Guo, "A scene-aware model adaptation scheme for cross-scene online inference on mobile devices," *IEEE Transactions on Mobile Computing*, vol. 24, no. 10, pp. 11 061–11 075, 2025.

[3] Y. Li, H. Zhu, Z. Deng, Y. Cheng, L. Zhang, S. Chang, and M. Guo, "Anole: Adapting diverse compressed models for cross-scene prediction on mobile devices," in *Proceedings of IEEE ICDCS*, 2024.

[4] S. Asadianfam, M. Shamsi, and A. Rasouli Kenari, "Big data platform of traffic violation detection system: identifying the risky behaviors of vehicle drivers," *Multimedia Tools and Applications*, vol. 79, no. 33, pp. 24 645–24 684, 2020.

[5] M. Nasr Azadani and A. Boukerche, "Driving behavior analysis guidelines for intelligent transportation systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 6027–6045, 2022.

[6] S. Liu, T. Wang, J. Li, D. Sun, M. Srivastava, and T. Abdelzaher, "Adamask: Enabling machine-centric video streaming with adaptive frame masking for dnn inference offloading," in *Proceedings of the 30th ACM International Conference on Multimedia*, ser. MM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 3035–3044. [Online]. Available: https://doi.org/10.1145/3503161.3548033

[7] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee, "The design and implementation of a wireless video surveillance system," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 426–438. [Online]. Available: https://doi.org/10.1145/2789168.2790123

[8] J.-Y. Wu, V. Subasharan, T. Tran, K. Gamlath, and A. Misra, "Mrim: Lightweight saliency-based mixed-resolution imaging for low-power pervasive vision," *Pervasive and Mobile Computing*, vol. 96, p. 101858, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574119223001165

[9] Y. Zhao, D. Luo, F. Wang, H. Gao, M. Ye, and C. Zhu, "End-to-end compression for surveillance video with unsupervised foreground-background separation," *IEEE Transactions on Broadcasting*, vol. 69, no. 4, pp. 966–978, 2023.

[10] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. 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*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 359–376. [Online]. Available: https://doi.org/10.1145/3387514.3405874

[11] V. Nigade, L. Wang, and H. Bal, "Clownfish: Edge and cloud symbiosis for video stream analytics," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, 2020, pp. 55–69.

[12] L. Zhang, H. Zhu, W. Fei, Y. Li, M. Zhang, J. Cao, and M. Guo, "Novas: Tackling online dynamic video analytics with service adaptation at mobile edge servers," *IEEE Transactions on Computers*, vol. 73, no. 9, pp. 2220–2232, 2024.

[13] L. Zhang, H. Zhu, Y. Li, J. Shen, and M. Guo, "The blind and the elephant: A preference-aware edge video analytics scheduler for maximizing system benefit," in *Proceedings of the 53rd International Conference on Parallel Processing*, 2024, pp. 317–326.

[14] Y. Wang, W. Wang, J. Zhang, J. Jiang, and K. Chen, "Bridging the edge-cloud barrier for real-time advanced vision analytics," in *Proceedings of the 11th USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'19. USA: USENIX Association, 2019, p. 18.

[15] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," 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*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 557–570. [Online]. Available: https://doi.org/10.1145/3387514.3405887

[16] J. Shen, H. Zhu, L. Zhang, Y. Li, S. Chang, J. Wu, and M. Guo, "Dive: Differential video encoding for online edge-assisted video analytics on mobile agents," in *Proceedings of IEEE ICDCS*, 2025.

[17] M. Viitanen, A. Koivula, A. Lemmetti, A. Ylä-Outinen, J. Vanne, and T. D. Hämäläinen, "Kvazaar: Open-source hevc/h.265 encoder," in *Proceedings of the 24th ACM International Conference on Multimedia*, 2016. [Online]. Available: http://doi.acm.org/10.1145/2964284.2973796

[18] A. Zhu, S. Zhang, X. Shi, K. Cheng, H. Sun, and S. Lu, "Crucio: End-to-end coordinated spatio-temporal redundancy elimination for fast video analytics," in *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, 2024, pp. 1191–1200.

[19] ISO/IEC, *High efficiency video coding*, International Organization for Standardization Std. 23 008-2:2013, 11 2013, mPEG-H Part 2, H.265.

[20] X. Jiang, P. He, T. Sun, and R. Wang, "Detection of double compressed hevc videos using gop-based pu type statistics," *IEEE Access*, vol. 7, pp. 95 364–95 375, 2019.

[21] Y.-T. Kuo, P.-Y. Chen, and H.-C. Lin, "A spatiotemporal content-based cu size decision algorithm for hevc," *IEEE Transactions on Broadcasting*, vol. 66, no. 1, pp. 100–112, 2020.

[22] C.-T. Ni, Y. Huang, and P.-Y. Chen, "A hardware-friendlyand high-efficiency h.265/hevc encoder for visual sensor networks," *Sensors (Basel, Switzerland)*, vol. 23, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257276758

[23] W.-J. Han, J. Min, I.-K. Kim, E. Alshina, A. Alshin, T. Lee, J. Chen, V. Seregin, S. Lee, Y. M. Hong, M.-S. Cheon, N. Shlyakhov, K. McCann, T. Davies, and J.-H. Park, "Improved video compression efficiency through flexible unit representation and corresponding extension of coding tools," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 12, pp. 1709–1720, 2010.

[24] J. Le Feuvre, C. Concolato, and J.-C. Moissinac, "Gpac: open source multimedia framework," in *Proceedings of the 15th ACM International Conference on Multimedia*, ser. MM '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 1009–1012. [Online]. Available: https://doi.org/10.1145/1291233.1291452

[25] FFmpeg Developers, "Ffmpeg." [Online]. Available: http://ffmpeg.org/

[26] HiSilicon. (2023) Hi3519av200 professional smart camera soc. Accessed: 2025-07-30. [Online]. Available: https://www.hisilicon.com/cn/products/smart-vision/pro-camera/Hi3519AV200

[27] G. Jocher and J. Qiu, "Ultralytics yolo11," 2024. [Online]. Available: https://github.com/ultralytics/ultralytics

[28] S. Chaudhary, A. Taneja, A. Singh, P. Roy, S. Sikdar, M. Maity, and A. Bhattacharya, "TileClipper: Lightweight selection of regions of interest from videos for traffic surveillance," in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. Santa Clara, CA: USENIX Association, Jul. 2024, pp. 967–984. [Online]. Available: https://www.usenix.org/conference/atc24/presentation/chaudhary

[29] M. Zhang, F. Wang, and J. Liu, "Casva: Configuration-adaptive streaming for live video analytics," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 2168–2177.

[30] K. Du, Q. Zhang, A. Arapin, H. Wang, Z. Xia, and J. Jiang, "Accmpeg: Optimizing video encoding for accurate video analytics," in *Proceedings of Machine Learning and Systems*, D. Marculescu, Y. Chi, and C. Wu, Eds., vol. 4, 2022, pp. 450–466.