TVM: A Tile-based Video Management Framework

Tianxiong Zhong
Beijing Institute of Technology
Beijing, China
inkosizhong@gmail.com

Zhiwei Zhang*
Beijing Institute of Technology
Beijing, China
zwzhang@bit.edu.cn

Guo Lu
Shanghai Jiao Tong University
Shanghai, China
sdluguo@gmail.com

Ye Yuan
Beijing Institute of Technology
Beijing, China
yuan-ye@bit.edu.cn

Yu-Ping Wang
Beijing Institute of Technology
Beijing, China
wyp_cs@bit.edu.cn

Guoren Wang
Beijing Institute of Technology
Beijing, China
wanggrbit@126.com

ABSTRACT

With the exponential growth of video data, there is a pressing need for efficient video analysis technology. Modern query frameworks aim to accelerate queries by reducing the frequency of calls to expensive deep neural networks, which often overlook the overhead associated with video decoding and retrieval. Furthermore, video storage frameworks optimize video retrieval through video partition or caching, often relying on prior information about the query workload. To further accelerate queries, this study introduces a novel tile-based video management framework, called TVM, which leverages the semantic information embedded in videos, without being dependent on specific query workloads. By constructing a tile-based semantic index for newly ingested videos, TVM effectively reduces the size of decoded and processed video data. To achieve this, TVM introduces an optimal index construction algorithm that utilizes cost function and pseudo-labels. Additionally, the framework proposes a query-driven tile parallel decoding algorithm and resource caching algorithms, which further expedite the retrieval of video frames. Experimental results demonstrate that TVM can significantly enhance the throughput of various query tasks, achieving a notable speedup of more than 5.6×.

PVLDB Reference Format:
Tianxiong Zhong, Zhiwei Zhang, Guo Lu, Ye Yuan, Yu-Ping Wang, and Guoren Wang. TVM: A Tile-based Video Management Framework. PVLDB, 17(4): 671 - 684, 2023.
doi:10.14778/3636218.3636224

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/InkosiZhong/TVM.

1 INTRODUCTION

With the widespread use of video capture devices, the amount of video data has been increasing rapidly. Meanwhile, the emergence of deep learning and computer vision has facilitated the development of applications for video analysis [24, 35, 50, 51, 56]. As a result, efficient video data analysis has gained significant attention from both industry and academia. To address the challenges posed by the massive video data and analysis demands, recent studies focus on efficient query processing [7, 24–26] and storage management [16, 18, 54], respectively. To accelerate the querying process, these works [7, 24–26] primarily concentrate on minimizing the number of invocations to computationally expensive object detection neural networks (NN). They achieve this through techniques such as model specialization and model cascading. However, these methods often assume that all video frames have already been decoded and stored in memory, thereby ignoring the overhead of video decoding. On the other hand, the storage frameworks [16, 18, 54] partition or cache videos based on the access frequency of video data in the query workloads, aiming to enhance data retrieval speed.

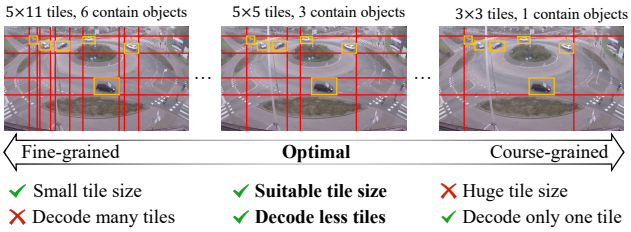


Figure 1: Optimal layout minimizes retrieval cost.

We introduce TVM, a tile-based video management framework, designed to efficiently accelerate various downstream queries. TVM leverages the inherent semantic information in videos to minimize the scale of data retrieved and processed, without being reliant on specific workloads. Specifically, we adopt a tile-based encoding and decoding scheme [16] and extract semantic information during video ingestion to construct an index that guides video encoding while accelerating the queries. Since TVM optimizes queries from a different perspective, it can be integrated with existing query and storage frameworks. However, the establishment of such a framework presents two main challenges.

C1. The construction of the semantic index is challenging. The extraction of embedded semantics from videos often necessitates the utilization of deep neural networks (DNNs), such as object detection DNNs. However, these DNNs are computationally expensive and contradict the optimization goals of query frameworks [7, 24–26]. Additionally, even after extracting the semantic information, the design of an optimal semantic index structure remains a crucial

*Corresponding Author.

Proceedings of the VLDB Endowment, Vol. 17, No. 4 ISSN 2150-8097.
doi:10.14778/3636218.3636224

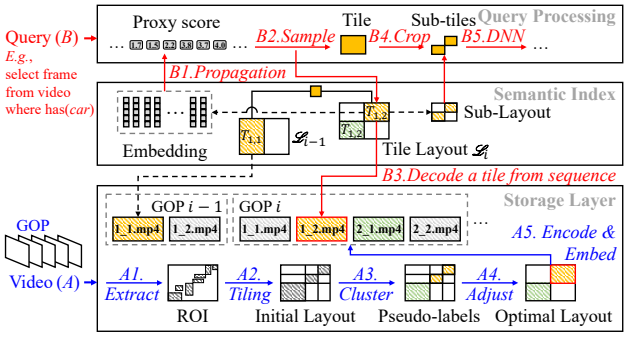


Figure 2: Pipelines of TVM. (A) Semantic index establishment. (B) Semantic index based query acceleration.

consideration. In TVM, the fundamental structure used to organize the semantic index and video data is the tile layout. Figure 1 shows the layouts with various levels of granularity based on the object position. The fine-grained layout consists of a large number of tiles, which leads to higher costs due to the initialization and destruction of decoders. On the other hand, the coarse-grained layout has a tile that encompasses a large number of pixels, resulting in slower decoding speeds. Therefore, finding the optimal layout that minimizes the overall costs becomes a significant challenge. ***C2.*** Effectively harnessing the semantic index and the structural features of tiles for query acceleration presents a complex challenge. Leveraging semantic indexing to filter out irrelevant pixels and reduce the size of decoding and inference operations can significantly enhance the performance of diverse queries. However, due to the varied nature of queries, their respective workflows for accessing tiles can differ significantly. In light of this, the design of query-driven acceleration algorithms becomes valuable and challenging.

To address *C1*, TVM employs lightweight algorithms to quickly extract semantic information from the video frames and generates an optimal layout by considering a cost function that accounts for various factors in tile-based video retrieval. Specifically, as depicted in Figure 2, the semantic index comprises a hierarchical tile layout and embedding vectors [27] corresponding to each tile within different tile sequences. This index is updated whenever new video segments are ingested, thereby guiding video encoding processes (Section 3). When a query is executed, TVM utilize the semantic index to identify the tiles that are related to the query and selectively decode and process those specific tiles (Section 4.1).

To tackle *C2*, we introduce a query-driven tile parallel decoding algorithm (Section 4.2) and several resource caching algorithms (Section 4.3). These components are specifically designed to leverage the access workflow of the queries, ensuring that the decoding process is tailored to the query's requirements.

In summary, our work makes the following contributions:

(1) We propose TVM, a novel tile-based video management framework that offers joint optimization of decoding and detection processes for diverse queries.
(2) We design a tile-based semantic index that efficiently accelerates queries without depending on query workloads.
(3) We develop a series of techniques to leverage the properties of tiles, thereby further enhancing the decoding process.

We evaluate the performance of TVM on various large-scale datasets using multiple query methods [7, 24, 26]. The experimental results demonstrate that TVM can significantly accelerate downstream query methods by more than 5.6×.

2 BACKGROUND

2.1 Video Encoding and Decoding

To reduce video storage overhead, video encoding methods [14, 44, 48] transform video frames from the pixel domain into a compressed bitstream. These methods divide the video into Groups of Pictures (GOPs), with each GOP starting with a keyframe (I-frame). The keyframes can be decoded independently, making them temporal random access points in video decoding. However, decoding other frames within the GOP requires dependence on previously decoded frames, typically forward-predicted frames (P-frames). Therefore, decoding a P-frame necessitates sequential decoding from the nearest keyframe. As keyframes require more bits for encoding, larger GOPs with larger keyframe intervals generally consume less bitrate but increase the delay for video random access.

The HEVC [48] decoder supports partitioning all frames in a GOP into non-overlapping tile sequences arranged in N rows and M columns. Each tile sequence is encoded individually, allowing for parallel decoding. Additionally, the use of tiles introduces spatial random access points for video decoding. The tiles belonging to the keyframes can be decoded independently, denoted as *keytiles*, while the decoding of other tiles needs to rely on the previously decoded tiles in the tile sequence. We describe the tile layout L in the form of a matrix, which indicates how the tiles are divided.

$$L = \begin{bmatrix} T_{1,1} & \dots & T_{1,M} \\ \vdots & \ddots & \vdots \\ T_{N,1} & \dots & T_{N,M} \end{bmatrix}, \tag{1}$$

where $T_{i,j} = (x_j, y_i, w_j, h_i)$ represents the offset and size of the tile at the i-th row j-th column. Specifically, $x_1 = y_1 = 0$, while $x_j = \sum_{l=1}^{j-1} w_l$ and $y_i = \sum_{l=1}^{i-1} h_l$ for $j > 1$ and $i > 1$, respectively. The change of layout occurs only between different GOPs, meaning the layout remains the same for all frames within a GOP. By utilizing homomorphic stitching [16, 19], all tile sequences can be packaged into a single container, enabling replay without re-encoding.

Certain methods [16, 18, 54] optimize the encoding format to accelerate video decoding. For instance, VSS [18] and TASM [16] cache or partition video based on the frequency of access to video data in the query workloads. Specifically, VSS caches copies with different coding formats for video segments to minimize the time required for video transcoding. TASM stores videos as non-overlapping tiles and adjusts the tile layout based on historical queries to enhance future sub-frame selection, which retrieves previously identified objects. VStore [54] proposes a backward derivation of configuration, modifying the fidelity and coding knobs for the video copies. These methods can leverage the query workloads to continually update their configuration. However, it is difficult for these methods to effectively utilize the semantic information within the video content before any query is executed. For example, TASM [16] only

provides a fine-grained initial tile layout around Regions of Interest (ROIs) when new video segments are ingested.

2.2 Analytic Query Processing

Video analytic queries aim to extract semantic information, such as object labels and locations from video frames. These queries typically involve decoding video frames into the pixel domain to analyze their content. Although some methods [6, 32, 33, 45] can extract semantic information from the encoded video stream without decoding, their accuracy is significantly lower compared to mainstream algorithms that operate in the pixel domain. Other studies [4, 5, 34, 46, 58] rely on hand-crafted features for extracting semantic information from videos, whose ability to deal with complex problems is limited by prior knowledge. Deep learning techniques have greatly advanced object detection, and object detection DNNs [12, 20, 23, 31, 41, 42, 49, 53, 57] have emerged as the dominant visual content analyzers for video analytic queries. Commonly studied query types include: ***Select* query** returns frames that satisfy a given query condition; ***Limit* query** imposes constraints on the quantity of returned records, *e.g.*, selecting *five* frames with cars; ***Aggregation* query** returns statistical information for a specific query object; ***Track* query** retrieves tracks that meet the specified query condition, *e.g.*, selecting car *tracks* that turn right.

Numerous methods [1, 2, 7, 11, 15, 24–27, 37] have been proposed to optimize the performance of various queries, with most of them focusing solely on the querying stage. These methods typically employ faster algorithms to generate prior information for guiding the selective processing of video frames by expensive DNNs. This approach eliminates the requirement for blind enumeration of all frames, resulting in a significant acceleration of the queries. For example, BlazeIt [24] trains specialized NNs to score each frame as a prediction of the DNN processing result, significantly speeding up the execution of *aggregation* queries and *limit* queries. SUPG [26] uses proxy scores to perform approximate *select* queries while guaranteeing statistical accuracy. TASTI [27] employs a pre-trained lightweight NN to embed video frames into a low-dimensional vector space, addressing the limitations of model specialization methods [24–26] that require separate NNs for different queries. MIRIS [7] eliminates redundant video information while avoiding uncertainty through NN-guided recursive interval sampling. These methods often assume that the video has already been decoded and cached in memory. However, with the decreasing cost of surveillance equipment, the number of cameras and their resolutions have significantly increased, which makes the assumption often untenable in reality. For instance, a single camera capturing 720p video at 30 frames per second (fps) generates over 20GB of video streams per day [54]. Storing all frames in the pixel domain and keeping them in memory would require terabytes (TB) of storage.

3 SEMANTIC INDEX ESTABLISHMENT

The semantic index comprises a tile layout, where each tile sequence that contains ROIs corresponds to a sub-layout and a set of embedding vectors corresponding to the tiles within it. As shown in the pipeline *A* in Figure 2, TVM generates the semantic index for each newly ingested GOP. The pipeline of indexing contains 5 steps from *A1* to *A5*. ***A1***, a lightweight background subtraction algorithm [3, 10, 13, 21, 36, 47, 55, 59, 60] is employed to generate ROIs, which are a series of bounding boxes without labels, providing approximate location and size information of the objects. These algorithms are computationally inexpensive and can be executed on smart cameras. ***A2***, to filter out irrelevant pixels, an initial tile layout is generated specifically around the ROIs. ***A3 & A4***, the optimal layout, which strikes a balance between decoding cost and decoder instantiation overhead, is determined by semantic-based layout adjustment algorithms (Section 3.3). ***A5***, the input frames are partitioned into the tile sequences based on the optimal layout, and each tile in the ROI-contained tile sequence will be embedded into a low-dimensional vector to extract the semantic information (Section 3.2). Meanwhile, to exclude irrelevant pixels from being input to the DNN, the initial layout and the optimal layout are combined to form a hierarchical layout (Section 3.4). Ultimately, the tile sequences are encoded as independent video streams (Section 3.5). It is worth mentioning that TVM organizes the index with the hierarchical schema, but uses the optimal layout of the outer layer to govern the encoding and decoding of tiles, while using the sub-layout of the inner layer to optimize DNN inferring, separately.

3.1 Preliminary Definitions

In this section, we first introduce the symbols used in our discussion. To avoid controversy, we use *tile region* to represent the position and size information of the tile, while using *tile* for the divided sub-frame in the following sections. $T_{i,j}$ denotes the i-th row and j-th column tile region in the layout. For the convenience of expression, it is also used to represent the tile sequence corresponding to the tile region. The GOP size is denoted as g, which is equal to the length of the tile sequences. Additionally, $T^t_{i,j}$ specifies a tile in the t-th frame among the whole video. The tile layout based on ROIs, consisting of N rows and M columns, is represented as $\mathscr{L}$. Specifically, compared to the trivial tile layout L, $\mathscr{L}$ additionally records whether each tile contains an ROI.

$$\mathscr{L} = \begin{bmatrix} T_{1,1}, r_{1,1} & \dots & T_{1,M}, r_{1,M} \\ \vdots & \ddots & \vdots \\ T_{N,1}, r_{N,1} & \dots & T_{N,M}, r_{N,M} \end{bmatrix}, \tag{2}$$

where $r_{i,j}$ is a Boolean variable indicates if $T_{i,j}$ contains any ROI.

TVM collects all ROIs within a GOP and utilizes them to generate the initial tile layout. Due to the limitations of the HEVC [48] codec, which does not support hierarchical layouts, a grid with dimensions $N \times M$ is generated to ensure that each ROI falls within a unique grid. The initial layout is created tightly around the boundaries of the ROIs, aiming to assign different ROIs to different grids and achieve a fine-grained layout [16].

Figure 1 illustrates different layouts with varying granularity, and they have different retrieval costs. The retrieval cost of a tile is denoted as $C(N_{pix}) = \mathcal{K} \times N_{pix} + \mathcal{B}$, which is positively correlated with the number of pixels N_{pix}. Assuming the GOP contains n different object labels that are scattered among tiles $O = \{o_1, o_2, ..., o_n\}$, the cost of each class is defined as $F_{\mathscr{L}}(o_l)$, which represents the time required to retrieve all tiles containing objects with label o_l.

$$\begin{aligned} F_{\mathscr{L}}(o_l) &= \sum C(h_i \times w_j), \text{ if } T_{i,j} \supseteq o_l \\ &= \mathcal{K} \times N^{total}_{pix} + m \times \mathcal{B}, \end{aligned} \tag{3}$$

where N_{pix}^{total} and m denote the total pixels and the number of tile regions containing o_l, respectively. Without loss of generality, the cost function $F_{\mathscr{L}}(O)$ is defined as the weighted average sampling time across all classes.

$$F_{\mathscr{L}}(O) = \sum_{l=1}^{n} \mathcal{W}_l \cdot F_{\mathscr{L}}(o_l), \sum_{l=1}^{n} \mathcal{W}_l = 1, \tag{4}$$

where $\mathcal{W}_l$ represents the weight associated with class o_l, which can be derived from the query workloads. Thus, $\mathscr{L}'$ is a better layout than $\mathscr{L}$, if $F_{\mathscr{L}'}(O) < F_{\mathscr{L}}(O)$. It should be noted that the fine-grained layout contains many tile regions, leading to high decoder initialization and release overhead. Intuitively, the m in Equation (3) will increase. On the other hand, the coarse-grained layout consists of only one tile region but contains a large number of pixels, *i.e.*, the N_{pix}^{total} will be huge, leading to high retrieval cost. The objective of TVM is to find an optimal layout that minimizes the cost function $F_{\mathscr{L}}(O)$, thereby reducing the retrieval overhead considering both the instantiation and decoding overheads.

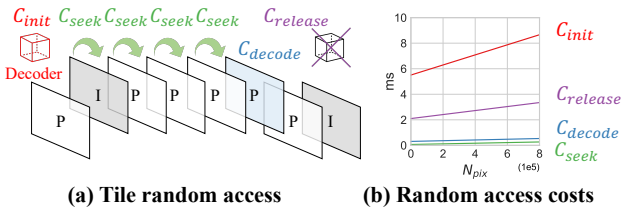
3.2 Embedding Generation

It has been extensively studied that the feature vectors produced by the hidden layer of image classification NNs trained on large-scale datasets, *e.g.*, ImageNet [17], can effectively represent the semantic information of an image [27]. These NNs efficiently non-linearly transform images and map them into low-dimensional feature spaces ($\mathbb{R}^n$), often referred to as embedding NNs. We follow TASTI [27] to fine-tune the embedding NN, which allows it to capture semantic similarities and differences more comprehensively. TVM employs the embedded NN in two stages. First, the embedding NN is harnessed to generate per-sequence embeddings. These embeddings are pivotal for clustering tile regions, a process that subsequently facilitates the creation of pseudo-labels (Figure 2-*A3*). Additionally, TVM creates per-tile embedding for each ROI-contained tile when indexing (Figure 2-*A5*). From the generated embedding vectors, a subset is selected as representative embeddings that have the maximum Euclidean distance between them, indicating significant semantic differences. TVM maintains a record of the closest-k representative embeddings to each embedding, along with their respective distances. In Section 4.1, we will describe how the cached information is utilized to calculate the proxy score during the propagation process [27].

3.3 Optimal Layout

To determine the optimal layout using the cost function $F_{\mathscr{L}}(O)$, it is necessary to define the retrieval cost $C(N_{pix})$. Since the video query frameworks [7, 25, 26] often reduce computing overhead through sampling, which involves random access to video frames, the retrieval cost $C(N_{pix})$ should be modeled as the cost of random access. Another key challenge is that the label set O is not known until the object detection DNN is called. However, it can be observed that the specific meaning of the labels in Equation (4) is not crucial. Therefore, an approximation of the label set O can be obtained by constructing pseudo-labels.

Tile random access. Frame random access refers to fetching frames in an order different from their encoded sequence. As illustrated in Figure 3a, when the video decoder searches for a specific



(a) Tile random access **(b) Random access costs**

Figure 3: The decoder performs random access decoding.

frame, it locates the nearest I-frame preceding the target frame and then decodes frames sequentially until the desired one is reached. The frames between the I-frame and the target frame are the dependent frames. The process of randomly accessing tiles is consistent with the frame random access. Specifically, The overhead associated with accessing the t-th tile in a tile sequence includes decoder initialization cost C_{init}, dependent tiles decoding cost $(t-1) \times C_{seek}$, target tile decoding cost C_{decode}, and decoder release cost $C_{release}$. The experiment demonstrates that any tile within the tile sequence is equally likely to be retrieved (Section 5.4). Therefore, on average, the decoder needs to decode $t = g/2$ tiles. The average sampling cost for a tile is modeled as the retrieval cost $C(N_{pix})$.

$$\begin{aligned}
C(N_{pix}) &= C_{init} + (\frac{g}{2} - 1) \times C_{seek} + C_{decode} + C_{release}, \\
C_{init} &= k_{init} \times N_{\text{pix}} + b_{init}, \\
C_{seek} &= k_{seek} \times N_{\text{pix}} + b_{seek}, \\
C_{decode} &= k_{decode} \times N_{\text{pix}} + b_{decode}, \\
C_{release} &= k_{release} \times N_{\text{pix}} + b_{release},
\end{aligned} \tag{5}$$

where the linear coefficients are dependent on the decoder and hardware configuration, and determined during initialization, the $\mathcal{K}$ and $\mathcal{B}$ equals the linear combination of them. Specifically, C_{init} includes the time required for the decoder to allocate memory space and preserve the CUDA context for GPU decoders, while $C_{release}$ includes the time to restore the CUDA context and release the memory. The difference between C_{seek} and C_{decode} is that C_{seek} only includes the time to decode the bitstream into frames with raw color formats, *e.g.*, YUV or NV12 color formats, while C_{decode} additionally contains the time to convert the image into RGB color format and further package it into the required data format, *e.g.*, array or tensor. As depicted in Figure 3b, C_{init} and $C_{release}$ are significant for the tile decoding and therefore cannot be neglected.

Pseudo-label Generation. Lines 1-8 in Algorithm 1 outline the process of constructing pseudo-labels based on semantic information. Specifically, for each ROI-contained tile sequence $T_{i,j}$, the tile that contains the largest ROI area is called a representative tile and is converted into the embedding $e_{i,j}$. Consequently, K embedding vectors are created, where K represents the number of ROI-contained tile regions in a layout. By computing the cosine similarity between embeddings, the tile regions can be clustered at a specified similarity threshold θ_{sim}. To organize the clusters, the union-find data structure P is utilized, where each element $P[i]$ in it corresponds to a parent node of the i-th tile region. Once the clustering is performed, the tile regions with the same parent are

Algorithm 1: Index Construction

Input: Layout $\mathscr{L}$, embeddings $E = [e_{i,j} | \text{if } r_{i,j}]$, threshold θ_{sim}
Output: Optimal layout $\mathscr{L}^*$

$P \leftarrow [1, 2, ..., |E|]$ // Union-find structure, "parents" list
// Cluster representative tiles with similar semantics
foreach $e[i] \in E$ **do**
  foreach $e[j] \in E - e[i]$ **do**
    if CosSimilarity$(e[i], e[j]) > \theta_{\text{sim}}$ **then**
      $p_i \leftarrow$ FindParent(i, P),
      $p_j \leftarrow$ FindParent(j, P),
      $P[p_i] \leftarrow p_j$; // Cluster in the same sub-tree
$\hat{O} \leftarrow$ ToPseudoLabels(P)
// Choose the layout with minimum cost
$\mathscr{L}^*, c^* \leftarrow \mathscr{L}, F_{\mathscr{L}}(\hat{O})$
foreach $\mathscr{L}' \in$ Spanning$(\mathscr{L})$ **do** // Up to 2^{N+M-2} layouts
  $c \leftarrow F_{\mathscr{L}'}(\hat{O})$
  if $c < c^*$ **then**
    $\mathscr{L}^*, c^* \leftarrow \mathscr{L}', c$
Return $\mathscr{L}^*$

considered to have the same pseudo-label, and therefore the P can be converted into $\hat{O}$ to approximate the real label set O.

Weight Generation. In the case of *unknown* workload, TVM simply set $\mathcal{W}_1 = \mathcal{W}_2 = \ldots = \mathcal{W}_n = \frac{1}{n}$, where $n = |\hat{O}|$. TVM will accumulate proxy scores corresponding to representative embeddings after each query is executed, and calculate weights when new video is ingested. Specifically, for the j-th query, the i-th representative embedding is initially converted into proxy score s_i^j, which serves as a metric for assessing the relevance of the current tile to the query (Section 4.1). For each representative embedding, TVM maintains a variable that accumulates the normalized proxy score $v_i = \sum_j (s_i^j / \sum_i s_i^j)$. Therefore, after establishing the pseudo-label, we obtain the closest-k representative embeddings for the embedding e_l corresponding to the parent node p_l and calculate the weight as Equation (6).

$$\mathcal{W}_l = \frac{e^{\hat{v}_l}}{\sum_{i=1}^{n} e^{\hat{v}_i}}, \hat{v}_l = \sum_{i'=1}^{k} v_{i'} \cdot \frac{\beta - d_{i'}}{\alpha}, \tag{6}$$

where $d_{i'}$ is the distance between e_l and the i'-th closest representative embedding, $\beta = \sum_{i'=1}^{k} d_{i'}$ and $\alpha = \sum_{i'=1}^{k} (\beta - d_{i'})$. Equation (6) assigns a larger weight to the content that is frequently queried, leading to a layout with lower $F_{\mathscr{L}}(\hat{o}_l)$ during index construction.

Index Construction. The key process of building the semantic index is to obtain the optimal tile layout. As shown in Figure 4, a row or column in the layout that does not contain any ROIs is referred to as *non-ROI*. In an ROI-based fine-grained layout, it is evident that rows or columns containing ROIs cannot be further divided, and subdividing the non-ROI rows or columns does not impact performance. Furthermore, resizing rows or columns without merging them leads to performance degradation. This is because the only possible adjustment is to expand the rows or columns containing ROIs while compressing the non-ROI ones, which introduces more irrelevant pixels into the tiles that need to be retrieved without reducing the number of tiles. Similarly, merging an

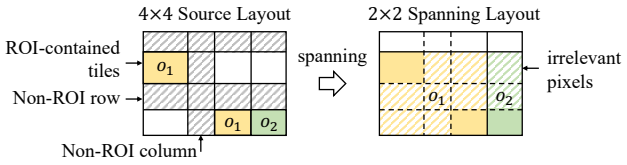


Figure 4: Spanning layout generation.

ROI-contained row/column with a non-ROI row/column adversely affects performance. Therefore, the optimized layout must be one of the spanning layouts, which involves merging adjacent rows and columns in the initial fine-grained layout. Moreover, both ends of the merged rows/columns must be ROI-contained rows/columns.

Algorithm 1 presents the overall procedure for constructing the optimal layout. By utilizing the pseudo-label $\hat{O}$ and the cost function $F_{\mathscr{L}'}(\hat{O})$, the cost of all spanning layouts generated from the initial layout can be computed. Specifically, the pseudo-label of a tile region after merging is the union of the pseudo-labels of the tile regions before merging. For example, merging two tile regions with pseudo-labels $\{o_1\}$ and $\{o_2\}$ respectively, will generate a new one with the pseudo-labels of $\{o_1, o_2\}$. Ultimately, the layout $\mathscr{L}^*$ with the lowest cost c^* is selected as the output optimal layout. Take Figure 4 as an example, TVM first creates an embedding for each ROI-contained tile region, *i.e.*, $T_{2,1}$, $T_{4,3}$ and $T_{4,4}$. Using the embeddings $E = [e_{2,1}, e_{4,3}, e_{4,4}]$, TVM cluster the tile regions resulting in $P = [1, 1, 3]$, which means $T_{4,3}$ has similar semantic with $T_{2,1}$ but different with $T_{4,4}$, leading to pseudo-labels $\hat{O} = \{o_1, o_2\}$. Next, TVM generates a 2×2 spanning layout $\mathscr{L}'$ by removing 2 vertical and 2 horizontal dividers from the initial layout. Through the cost function $F_{\mathscr{L}'}(\hat{O})$, TVM calculates and records the cost of the layout, and evaluates the next spanning layout.

Efficient Index Construction Procedure. For a given layout with N rows and M columns, it can be generated by cutting a rectangle with $N-1$ horizontal and $M-1$ vertical dividers. Generating a spanning layout involves removing some of the dividers, resulting in up to 2^{N-1} possible combinations of rows and 2^{M-1} possible combinations of columns. Therefore, the time complexity of traversing all spanning layouts is approximately $O(2^{N+M})$.

It is worth noting that if the rows (columns) of the layout are determined, searching for the best combination of columns (rows) becomes an interval dynamic programming (DP) problem. Given a layout $\mathscr{L}$ with fixed rows, the optimal solution for the first i columns is independent of the subsequent columns. Given an anchor $j < i$, keep the optimal solution of the first j columns and merge columns from $j+1$ up to i to form a candidate solution. The optimal solution must belong to the set of candidate solutions. Therefore, the goal is to find the optimal anchor j, which can be expressed using the following DP recurrence relation:

$$dp[i] = \min_{1 \le j < i} (F_{\mathscr{L}[:,1:i]}(\hat{O}), dp[j] + F_{\mathscr{L}[:,j+1:i]}(\hat{O})), \tag{7}$$

where $dp[i]$ and $dp[j]$ are the minimum cost of the first i and j columns, respectively, and $F_{\mathscr{L}[:,j+1:i]}(\hat{O})$ represents the cost of the merged column. The optimal column layout can be obtained through a double loop, resulting in a complexity of $O(M^2)$ for the column dimension, and the total complexity is $O(2^N \cdot M^2)$. By

executing the DP algorithm on the larger dimension, the overall complexity can be controlled at $O(2^{\min(N,M)} \cdot \max(N,M)^2)$.

However, in cases where both N and M are large, a greedy algorithm is proposed to reduce the time complexity to $O(K^3)$ and still yield an optimal layout in most scenarios. Specifically, the algorithm selects a pair of ROI-contained tile regions, T_{i_1,j_1} and T_{i_2,j_2}, from the current layout, and merges all the rows and columns between the tiles to generate the spanning layout. Subsequently, it chooses the least cost layout among all spanning layouts. The tile regions are recursively merged until merging any pairs no longer reduces the cost. Considering the existence of non-ROI rows and columns, we have the range $\max(\frac{N-1}{2}, \frac{M-1}{2}) \leq K \leq N \cdot M$. In most cases, the ROI-contained tile regions are sparsely distributed, making the greedy algorithm an effective approach to accelerate the adjustment. In summary, TVM employs different strategies depending on the complexity of the layout. In most cases, it utilizes the DP algorithm. When both N and M are large, TVM resorts to the greedy algorithm, enabling a more efficient indexing process.

3.4 Hierarchical Layout

Only the decoder aspect is taken into consideration during the search process for the optimal layout, without accounting for the object detection DNNs utilized by the query. The reason behind this is that different query tasks may employ different DNNs, and optimizing the tile layout for a specific DNN would render it unsuitable for others. To enhance the inference performance of DNNs, a hierarchical layout is generated by combining the initial layout with the optimal layout. The tile regions present in the initial layout form sub-layouts within the tile regions of the optimal layout. For instance, as shown in Figure 2, $T_{1,2}$ corresponds to a 2×2 sub-layout, where only 2 *sub-tiles* contain the ROIs. Again, the optimal layout is employed for video encoding and decoding purposes, while the sub-layout is utilized to optimize DNN inference (Section 4.1).

3.5 Tile-based Video Storage

The incoming video frames are divided based on the specified optimal layout, and the tile sequences are encoded into video bitstreams. Furthermore, all tile sequences belonging to the same GOP are organized in a common folder. When processing a given input video stream, users have the option to determine the bitrate, which refers to the number of bits consumed to encode the frames within one second. TVM ensures that storing the tile sequences does not require additional space by allocating the bitrate for each tile based on the proportion of pixels it contains compared to the original untiled video frame. The Constant Bit Rate (CBR) strategy is employed to control the encoding process.

4 SEMANTIC INDEX BASED ACCELERATION

As shown in the pipeline B in Figure 2, the semantic index can be used to accelerate the query processing and combined with the existing works. The pipeline contains 5 steps from *B1* to *B5*. ***B1***, TVM derives proxy score [24, 26, 27] from the embeddings in the semantic index using a user-defined scoring function. The process of generating the proxy score is referred to as *Propagation* [27]. ***B2***, the query frameworks [7, 24–26] selectively sample the tiles based on the proxy score. ***B3***, given a ternary coordinate (t, i, j) from the query frameworks, the $T^t_{i,j}$ is retrieved from the tile sequence $T_{i,j}$ belonging to the $\lfloor t/g \rfloor$-th GOP. ***B4***, the sub-layout associated with $T_{i,j}$ is also retrieved from the semantic index. A similar layout adjustment algorithm is then employed to search for the optimal sub-layout for DNN inference. ***B5***, by cropping the tiles based on the sub-layout, sub-tiles can be generated, effectively reducing the number of irrelevant pixels that are fed into the DNN. This process contributes to further accelerating the inference stage. The index offers a query-independent acceleration for TVM, enabling the reduction of pixel scale for processing (Section 4.1). Further, by analyzing the sampling workflows of different queries, we provide query-driven acceleration methods in Section 4.2 and Section 4.3.

4.1 Query-independent Acceleration

As mentioned in Section 2.2, several methods [24–26] utilize proxy scores as a means to approximate the outputs of resource-intensive DNNs. We adopt the approach introduced in TASTI [27] to generate proxy scores for each tile. Specifically, all tiles associated with representative embeddings are detected by the object detection DNNs. The user or query frameworks provide a scoring function that specifies how to quantify the output of object detection DNNs into proxy scores. For instance, a scoring function for a vehicle *aggregation* query would take records from the DNNs as input and produce the count of vehicles as output. Furthermore, the proxy scores are propagated from the tiles corresponding to the representative embeddings to all ROI-contained tiles. To generate the proxy score s from an embedding, the distances to its closest-k representative embeddings d_i are normalized to weight the proxy scores s_i of the corresponding representative embeddings, $s = \sum_{i=0}^{k} s_i \cdot \frac{\beta - d_i}{\alpha}$, where $\beta = \sum_{i=1}^{k} d_i$ and $\alpha = \sum_{i=1}^{k} (\beta - d_i)$ are the normalization parameters, converting distance d_i into weight. Moreover, non-score-based query frameworks [2, 7, 30, 37] can also benefit from acceleration using embeddings by defining a default scoring function that outputs the number of query objects. If the query framework samples a frame, all zero-scored tiles within the frame can be pruned and the rest of the tiles are considered to contain the query target.

By using the optimal layout and embeddings, TVM effectively excludes irrelevant pixels from the decoding process. However, since the optimal decoding layout may introduce more irrelevant pixels compared to the initial layout, this can have a negative impact on DNN inference. To address this issue, a hierarchical layout is introduced in Section 3.4. This approach divides a tile with the sub-layout, which records the sub-tile regions that are irrelevant. Specifically, when a tile is decoded, its corresponding sub-layout is retrieved. Similarly, like the optimal layouts for decoding, there are also optimal sub-layouts for a specified DNN that minimize the inference time. Modern object detection DNNs [12, 23, 31, 41, 49] often allow users to adjust the size of the input image as a trade-off between accuracy and throughput. Considering that some models will limit the minimum and maximum size of the input image, the inference cost $C^{nn}(N_{pix})$ can be modeled with a piecewise function,

$$C^{nn}(N_{pix}) = \max(\min(k_{nn} \times N_{pix} + b_{nn}, c_M), c_m), \quad (8)$$

where k_{nn} and b_{nn} are linear coefficients that depend on the hardware and DNN architecture, c_m and c_M are costs when inference on frames with the lower and higher size bounds, respectively. These

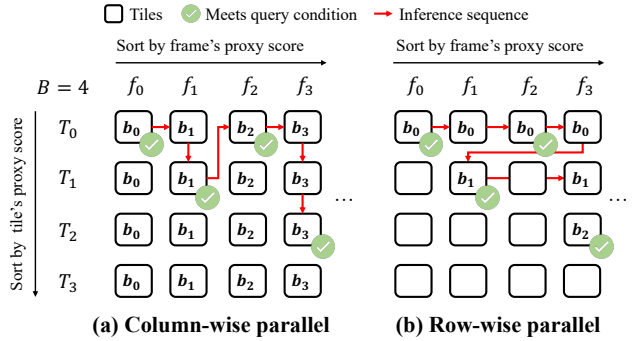


Figure 5: Row-wise parallel strategy effectively avoids decoding waste for *limit* query.

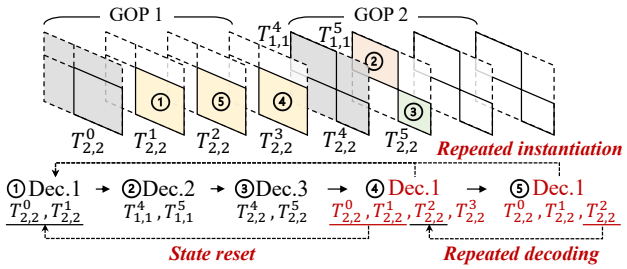


Figure 6: Retrieval tiles from rank 1 to 5, displaying the utilized decoders, accessed tiles, and potential inefficiencies.

coefficients are fitted by inferring various sizes of all-zero tensors. To avoid sub-layout adjustments becoming a bottleneck, it is assumed that different sub-tile regions have the same pseudo-label. Therefore, the cost function $F^{nn}_{\mathscr{L}}$ of the sub-layout $\mathscr{L}$ is simplified as the sum of the costs of all sub-tile regions containing the ROIs.

$$F^{nn}_{\mathscr{L}} = \sum C^{nn}(h_i \times w_j) \text{ if } r_{i,j}. \tag{9}$$

TVM omits the semantic clustering step in the layout adjustment algorithms (Section 3.3) and instead executes the same spanning layout searching algorithm, depending on the cost function $F^{nn}_{\mathscr{L}}$, to obtain the optimal sub-layout. Subsequently, as shown in Figure 2, the ROI-contained sub-areas are cropped from the decoded tiles, generating sub-tiles. These sub-tiles are then input into the DNN sequentially, and the results are summarized.

4.2 Query-driven Parallel Tile Decoding

Different queries employ various frame sampling strategies, resulting in different workflows. For example, BlazeIt [24] performs uniform random sampling for *aggregation* queries. When conducting *limit* queries, video frames are sorted based on the proxy scores, and sampling is performed from lower-ranked tiles. SUPG [26] employs non-uniform random sampling weighted by the proxy scores. MIRIS [7] employs built-in GNN and RNN to determine whether to perform supplementary sampling. When deploying these frameworks on tile sequences, *aggregation* queries, *select* queries, and *track* queries have no dependencies among different tiles within a frame. This means that the sampling of the next tile does not rely on the processing result of the current tile and the sampling workflow can be generated in advance (*known*).

However, there is a dependency between tiles in *limit* queries, which makes the workflow *unknown*. As shown in the Figure 5, to conduct *limit* queries on a tile-based schema, all tiles must be sorted and organized in a 2-D grid. Each column of elements represents tiles $[T_0, T_1, ...]$ belonging to the same video frame f_i, and the frame order is determined by the sum of the proxy scores of all tiles within the frame. The red arrow in Figure 5a illustrates an intuitive serial sampling sequence where sampling is first performed by column until the retrieved tiles meet the query condition, and then sampling continues from the first tile in the next column. Consider a *limit* query, *Select 4 frames with at least 2 cars*. T^0 of f_0 contains 2 cars, therefore meeting the query condition. Subsequently, as the retrieval workflow progresses to the next frame f_1, both T_0 and T_1 contain 1 car each. The workflow then shifts to the subsequent frames in a similar manner. The column-wise parallel strategy directly follows the sampling priority in the serial sampling scheme to enable parallel decoding. Specifically, b_i in Figure 5 represents the i-th batch of retrieved tiles. In this approach, many tiles are decoded but not inferred, *e.g.*, the T_1, T_2, and T_3 in f_0, leading to unnecessary decoding. To address this issue, the row-wise parallel strategy, shown in Figure 5b, is employed. In this strategy, the grid is treated as a 2-D queue. When a frame meets the query condition, the entire column corresponding to that frame is popped out from the queue. Each time, the first B tiles in the first row of the queue are popped for decoding. Through row-wise parallelism, tiles that do not require inference are not decoded.

Considering that different query tasks follow distinct retrieval workflows, TVM adopts a query-driven parallel decoding schema. For queries without tile dependencies, TVM conducts parallel decoding with a batch size of B according to the *known* workflows. Furthermore, for *limit* queries, TVM employs the row-wise parallel strategy, as it effectively avoids decoding unnecessary tiles.

4.3 Query-driven Resource Caching

The retrieval workflow of certain queries, *e.g.*, the *track* queries, involves sequential access to video data at specific sampling intervals, and the sampling between intervals exhibits a strong positional dependence. Treating this type of workflow as pure random access without optimization will consume much time to initialize and release the decoder, as well as repeated frame decoding. Specifically, three types of inefficiencies can be identified: **Repeated instantiation of decoders.** When the same tile sequence is accessed repeatedly, the decoder is constantly created and destroyed. As depicted in Figure 6, in the case of sequential retrieval of tiles $T^1_{2,2}$, $T^5_{1,1}$, $T^5_{2,2}$, $T^3_{2,2}$ and $T^2_{2,2}$ within the tile sequence, decoder 1 must be instantiated three times. **State reset.** When the target tile is a successor to the decoded tile in the same tile sequence, the default implementation still returns the most recent keytile. For instance, assuming that decoder 1 is not destroyed, when decoding the tile $T^3_{2,2}$, the decoding process starts from $T^0_{2,2}$, discarding the progress made on $T^1_{2,2}$. **Repeated decoding.** Similar to the state reset issue, the default implementation returns to the most recent keytile when decoding a target tile that is a predecessor of an already decoded tile,

Table 1: Video Datasets

Dataset	Resolution	Duration	Query objects
night-street	1920×1080	27 hours	car, person
amsterdam	1280×720	33 hours	car, person, bicycle
archie	3840×2160	33 hours	car, person, bicycle
canal	1920×1080	18 hours	boat

even though the target tile has been previously decoded. For example, When decoding $T^2_{2,2}$, it initiates decoding from $T^0_{2,2}$, despite the fact that $T^2_{2,2}$ has already been decoded. To address these problems, several caching optimization strategies can be implemented:

Codec caching. To mitigate the repeated instantiation issue, decoders that have already been created are cached, and a least recently used (LRU) algorithm is employed to manage the cache. When new access occurs, the cache is checked for a decoder associated with the same physical file. If the decoder is found, it is fetched and used for decoding; otherwise, a new one is created. Once the tile is decoded, the decoder is cached as an object. When the cache is full, the LRU decoder is removed from the cache and destroyed. **State caching.** By caching the decoding states of used decoders, the problem of state reset can be addressed. During tile access, if the decoder is accessible in the cache, the current decoding position is evaluated. When the target tile is a successor to the currently decoded tile, decoding can continue from the current position without returning to the most recent keytile. **Tile caching.** In addition to codec caching, caching the decoded tile can be beneficial. During the access, the dependent tiles (predecessors of the target tile) are also decoded into their original color space, *e.g.*, YUV or NV12. The raw-format tiles can be cached and quickly converted to widely used RGB color format and suitable data formats (*e.g.*, tensor) when accessed. TVM uses the LRU algorithm for tile caching.

5 EVALUATION

We implemented a prototype of TVM in Python. In our implementation, TVM utilizes FFmpeg [9] with the HEVC [48] codec for tile encoding. To evaluate TVM, we employ the HEVC decoder implemented by the Video Processing Framework [39], which is a Python wrapper of Nvidia Video Codec [38]. The experiments are conducted on a single node running Ubuntu 20.04, equipped with an Intel i9-9900KF processor and an Nvidia RTX2080Ti GPU.

We evaluate TVM on videos of various resolutions and content, as detailed in Table 1. These datasets are commonly used in video analytics evaluations [15, 24–27, 37] and range in resolution from 720p (HD) to 2160p (4K). Following the approach of TASTI [27], we employ ResNet-18 [20] to generate 128-dimensional embedding vectors. Additionally, we utilize YOLOv5 [23] as the object detection DNN. We use *Frame* to represent our baseline method, TASTI [27], which builds frame-based semantic embedding on the untiled videos. To validate the superiority of our semantic index, we also conducted a comparative analysis of another tile-based video storage manager, TASM [16].Moreover, we introduced a variant denoted as *TASM+*, which incorporates the identical tile parallel decoding algorithm proposed by TVM. All sequences employ a GOP size of 30 to ensure low latency for random access.

To demonstrate the effectiveness of TVM, we use BlazeIt [24] for *aggregation* and *limit* queries, SUPG [26] for approximate *select* queries, and MIRIS [7] for *track* queries. The objects to be queried are shown in Table 1. For *aggregation* queries, we evaluate TVM by counting the number of objects with a 0.01 error tolerance and a success probability of 95%. In *limit* queries, we select 1,000 frames containing the target object. When evaluating TVM on SUPG, we set a detection DNN budget of 10,000 and a recall target of 90%. For *track* queries, we employ MIRIS to generate the retrieval workflows and test the decoding and detection overhead offline. The ground truth of the tracks is obtained using the YOLOv5 [23] frame-by-frame and DeepSORT [52] tracking algorithm.

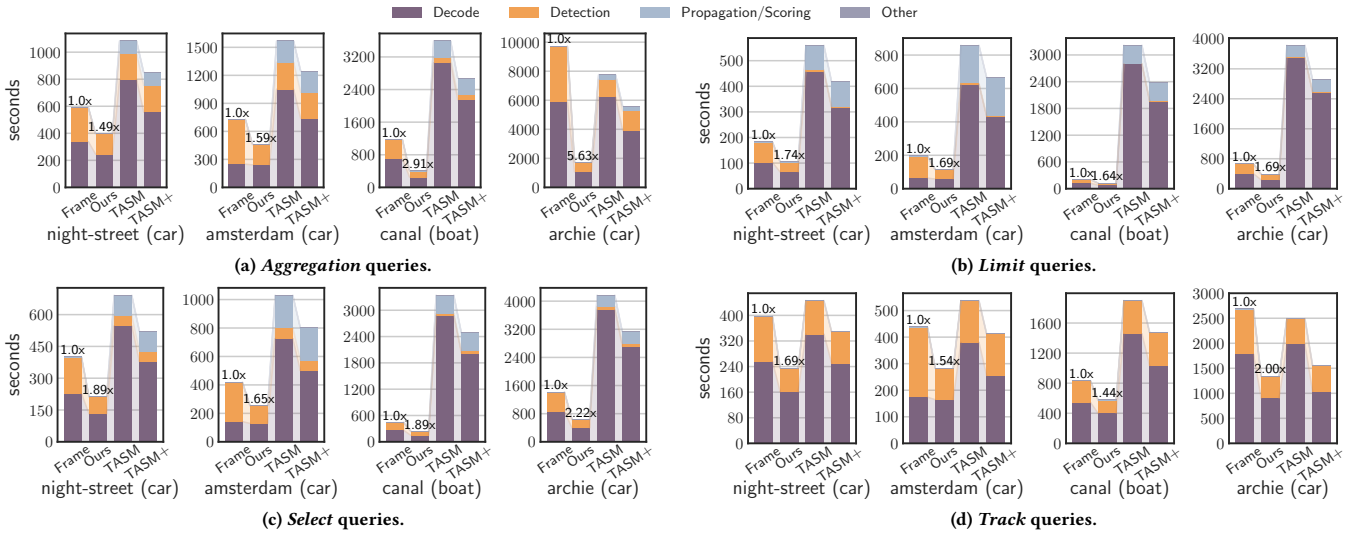
We present four types of costs. **Propagation/Scoring** involves calculating the proxy score for each frame. Propagation utilizes the embedding to generate the proxy scores. Since it is independent for each frame, we use numba [29] to speed up the process in parallel. In contrast, scoring directly employs the lightweight specialized NNs [24–26] for online score generation. In our experiments, we use TRN10 [24] as the specialized NN. **Decode** includes the entire cost of retrieving data, including random access during the query process and sequential access during the scoring process (*TASM* and *TASM+*). **Detection** represents the time-consuming use of the object detection DNN to extract objects from the retrieved video frames. It is noteworthy that the decoding and detection costs brought about by the propagation process for the tiles corresponding to the representative embeddings (Section 4.1) are also calculated for a fair comparison. **Other** encompasses time overhead apart from propagation, decoding, and detection, *e.g.*, the time required for post-processing the results obtained from the DNN.

5.1 Query Acceleration

Figure 7 presents a comparison of the time consumption spent by different methods at each stage. Our method outperforms the baseline methods across all datasets, achieving a notable speedup of 5.6× compared with the frame-based method. Due to space constraints, the query performance for more different query objects is shown in Table 2. In the following analysis, we will examine the experimental results based on different types of queries.

Aggregation query. According to Figure 7a, TVM achieves a maximum speedup of more than 5.6× on the archie dataset compared with the frame-based method, saving about 82% of both decoding and detection time. When querying on the night-street and canal dataset, TVM saves 27% and 64% of the decoding time, and 40% and 69% of the detection time, respectively. For the amsterdam dataset, although TVM shows a slight advantage in the decoding stage, it maintains a high speedup ratio in the detection stage, resulting in a 54% overhead reduction. This indicates that even though TVM's support for low-resolution videos may not be as strong as for high-resolution ones, it still provides significant acceleration for queries. In contrast, TASM [16] uses the specialized NNs online to score each tile, leading to high decoding and scoring overhead. Meanwhile, TASM+ reduces the decoding overhead by 32% on average through parallel decoding, proving the effectiveness and transferability of our approach.

Limit query. TVM outperforms the baseline methods, achieving an average speedup of up to 1.7× across different datasets, as



(a) *Aggregation* queries.

(b) *Limit* queries.

(c) *Select* queries.

(d) *Track* queries.

Figure 7: Time consumption of queries against *car* and *boat* objects for different methods.

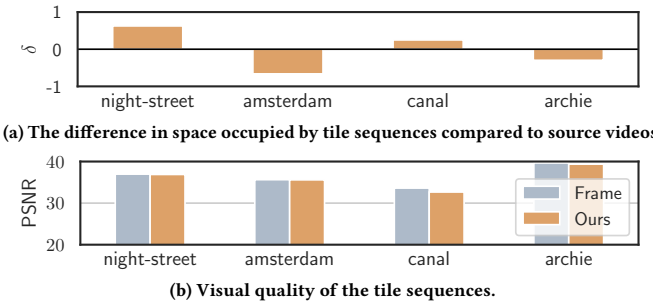
Table 2: Time consumption (s) of queries (Q.) against *person* and *bicycle* objects for different methods (M.).

M. \ Q.	night-street *person*				amsterdam *person*				amsterdam *bicycle*				archie *person*				archie *bicycle*			
	agg.	*limit*	*select*	*track*	*agg.*	*limit*	*select*	*track*	*agg.*	*limit*	*select*	*track*	*agg.*	*limit*	*select*	*track*	*agg.*	*limit*	*select*	*track*
Frame	1029	184	400	297	4574	198	424	724	888	196	421	442	3298	663	1410	2269	1739	663	1405	2521
TASM	1075	552	692	456	3513	857	1014	565	1155	1121	1001	540	5887	3844	3988	2480	4633	3983	4039	2302
TASM+	816	411	529	348	3041	664	792	447	939	880	776	410	4443	2868	3013	1535	3518	3012	3048	1536
Ours	**494**	**103**	**212**	**134**	**2357**	**119**	**251**	**442**	**467**	**118**	**257**	**231**	**1845**	**391**	**569**	**1391**	**1107**	**396**	**593**	**1032**

illustrated in Figure 7b. Specifically, on the archie dataset, TVM can save more than 41% and 42% of the time during the decoding and detection stages, respectively. Similar results can be observed on the night-street and canal datasets. For the amsterdam dataset, although the speedup of TVM in the decoding stage is relatively small (8%), it still achieves a time overhead reduction of more than 58% in the detection process. Since TASM's high scoring cost, it accounts for a high proportion of the overhead of *limit* queries.

Select query. When evaluating TVM on SUPG queries [26], the use of tiles in TVM with smaller resolutions can significantly accelerate the sampling stage. As depicted in Figure 7c, on high-definition videos such as the night-street, archie, and canal datasets, TVM achieves an average speedup of 2×. Particularly on the archie dataset, TVM can save more than 55% and 56% of the time during the decoding and detection stages, respectively. Similarly, TASM is constrained by online scoring, resulting in high overhead.

Track query. As shown in the Figure 7d, TVM demonstrates superior performance compared to the baseline method by achieving a 1.7× average speedup. Specifically, our method can save more than 37%, 5%, 26%, and 49% of decoding time on the four datasets, respectively. Moreover, the proportions of time saved in the detection stage are 47%, 55%, 39%, and 51%, respectively. Since MIRIS [7] does not use proxy score filtering, TASM [16] no longer requires high scoring overhead. However, without the guidance of the proxy score, TASM is compelled to retrieve all tiles containing ROIs in the specified frames, even if some of these tiles might not be pertinent



(a) The difference in space occupied by tile sequences compared to source videos.

(b) Visual quality of the tile sequences.

Figure 8: Storage overhead and visual quality.

to the query object. For example, in the canal dataset, numerous tiles containing ROIs are not indicative of *boat* presence; rather, they are attributed to false detections stemming from water wave movements. Coupled with TASM's use of fine-grained layouts, it ultimately leads to higher decoding and detection consumption.

5.2 Storage Overhead and Visual Quality

As highlighted in Section 3.5, TVM employs the area of tile region to regulate bitrate allocation. Figure 8a illustrates that our method allows for precise control of the storage overhead, maintaining a relative error (δ) of no more than 1% across all datasets. To assess

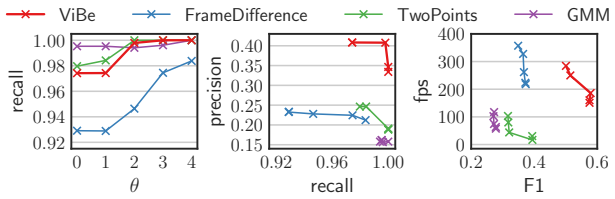
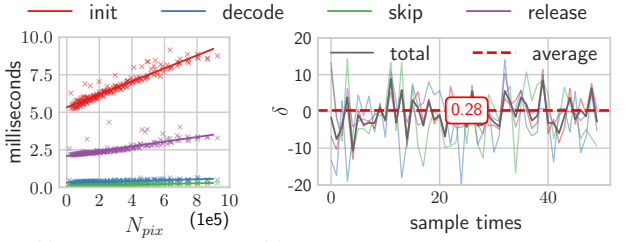


Figure 9: Quality of ROI generated by various background subtraction algorithms and distance threshold θ.



(a) Coefficients fitting. **(b) Relative error in predicting time of random access.**

Figure 10: The validity of the cost function.

the visual quality of tile sequences, we separately encode untiled videos and tile sequences using the same bitrate. Subsequently, we decode and concatenate the tiles and calculate Peak Signal-to-Noise Ratio (PSNR) with the original frames. As depicted in Figure 8b, the tile sequences exhibit a visual quality that is very close to the untiled videos across all datasets. Generally, when the PSNR exceeds 30dB, the differences between images are barely discernible.

5.3 Layout Fidelity

A high-fidelity tile layout aims to minimize the loss of objects, ensuring that the downstream queries are error bounded. TVM addresses the challenge by consolidating the ROIs generated by the background subtraction algorithm. These ROIs are merged if their normalized mutual distance falls within a threshold θ. Specifically, the normalized distance is a ratio of the center distance of two ROIs to the minimum of their diagonal lengths. Figure 9 illustrates the impact of various widely used background subtraction algorithms, *i.e.*, ViBe [3], FrameDifference, TwoPoints and GMM [59], and thresholds on layout fidelity. In this context, *recall* signifies the proportion of retained objects, *precision* denotes the area ratio of objects' bounding boxes within the tile, and *fps* is the throughput of generating ROIs. The results indicate that the ViBe algorithm [3] achieves the highest precision and considerable throughput, while almost losing no objects. Thus, we have adopted the ViBe algorithm with a threshold value of $\theta = 2$ as the default configuration.

5.4 Cost Function and Random Access

During the system initialization of TVM, the decoder-based and hardware-based coefficients of the cost function are determined. This process involves resizing a preset short video (consisting of only 150 frames) to various resolutions ranging from 130×130 to 1920×1080 and encoding them into HEVC [48] video streams. Subsequently, each video stream is decoded to measure the corresponding time cost, and the number of pixels (N_{pix}). Finally, a least square method is used to fit the coefficients, which are then saved as metadata of the system. As depicted in Figure 10a, the linear functions can effectively model the overhead of the decoder. To demonstrate the accuracy of Equation (5), an evaluation is conducted to predict the time of random access on the tiled canal dataset. A series of tiles are selected from the pre-processed dataset, and the predicted C_{init}, C_{seek}, C_{decode} and $C_{release}$ are compared with the actual processing times. As illustrated in Figure 10b, the total error per sample is represented by the blue line, and the average error over 50 samples is shown as the red dashed line. The error of a single sampling does not exceed 10%, while the average error is only 0.28%. In order to validate the average decoding tiles in Equation (5), we conducted four queries using the archie dataset. As shown in Figure 11, the sampling distribution for *aggregate*, *limit*, and *select* queries closely adheres to uniform sampling. Due to the non-fully random access nature of *track* query, its sampling pattern exhibits a certain degree of periodicity. Nevertheless, across all query types, the average count of sampled tiles remains remarkably close to the assumed value of $g/2$ which confirms the plausibility of the hypothesis. The results indicate that Equation (5) reasonably describes the cost of large-scale random access and can be relied upon for accurate predictions.

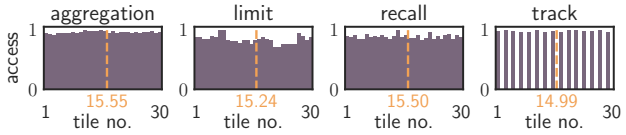
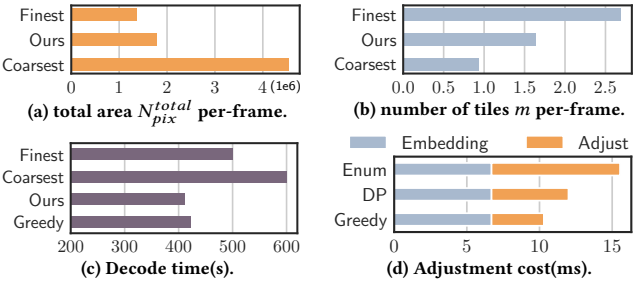


Figure 11: The normalized frequency of accessing each tile in the tile sequence by different queries.



(a) total area N_{pix}^{total} per-frame. **(b) number of tiles m per-frame.** **(c) Decode time(s).** **(d) Adjustment cost(ms).**

Figure 12: Layout comparison of different granularities.

5.5 Effectiveness of Layout Adjustment

To fairly verify the effectiveness of the layout adjustment algorithm, we conducted a comparative analysis with the fine-grained and coarse-grained layouts proposed by TASM [16] using an identical sub-frame selection query. This query aims to minimize the time required to retrieve pixels containing the *car* objects, that are previously identified. TASM [16] points out that fine-grained layout

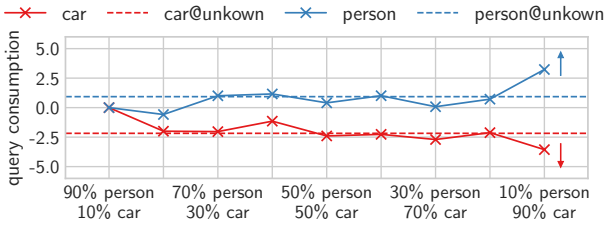


Figure 13: The variation trend of query consumption under different workloads.

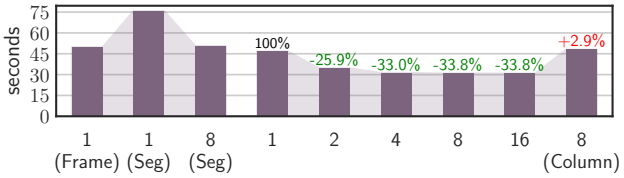


Figure 14: Compare the decoding time of a *limit* query for different numbers of threads and parallel methods.

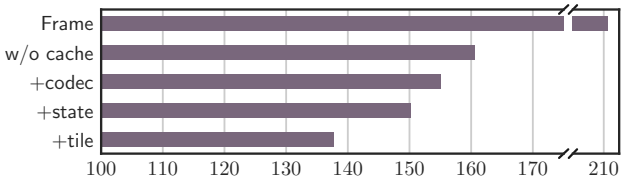


Figure 15: Compared the decoding time (s) of a *track* query when the caching algorithms are gradually added.

outperforms coarse-grained layout. However, both of them may not be optimal. In the experiment conducted on the archie dataset, the YOLOv5 [23] object detection DNN is called frame by frame to generate the two baseline layouts as well as the adjusted layouts. Instead of using semantic embeddings in the clustering stage, the real object labels are used, and the same layout spanning algorithms are applied to find the least-cost layout.

Figure 12a and Figure 12b show that the adjusted layout generated by TVM represents an intermediate granularity between the baseline layouts, with the total area and the number of tiles falling between them. Figure 12c demonstrates that the adjusted layout outperforms both baseline layouts. Compared to the fine-grained and coarse-grained layouts, our approach can save approximately 18% and 32% of decoding time, respectively, indicating the effectiveness of our layout adjustment algorithm. Additionally, the performance of the sub-optimal layout generated by the greedy algorithm only drops by less than 3% compared to the optimal layout. The executing efficiency of the proposed layout adjustment algorithms is evaluated in Figure 12d. Both the dynamic programming (DP) algorithm and the greedy algorithm significantly improve the throughput of the layout adjustment algorithm. It is important to note that the layout adjustment algorithm is executed only once per GOP during video ingestion and does not require generating embeddings during sub-layout adjustment, making it computationally efficient.

Finally, we conduct experiments to show the performance of TVM under different workloads. By adjusting the weights in Equation (4), TVM optimizes the layout to decrease the retrieval cost of frequently queried objects, affecting subsequent query performance. We show the average performance of four queries on different workloads in Figure 13. As the query frequency of *car* in the workload increases, the efficiency of querying *car* becomes higher, while the efficiency of querying *person* decreases. Meanwhile, we also show the query performance under *unknown* workload for intuitive comparison. When comparing its performance to the *known* workloads, the variance does not exceed ±3%. This makes it acceptable to make layout adjustments based on the assumption of uniform distribution for workloads.

5.6 Effectiveness of Query-driven Acceleration

Since the Video Processing Framework[39] utilizes GPU for achieving high parallel decoding within a frame or tile, our optimization considerations are focused on parallelization between frames or tiles. In Figure 14, we present the decoding consumption for the *limit* query on the archie dataset. To account for scenarios where videos are stored in segments, we also exhibit the decoding performance on segmented untiled videos. On the left side of Figure 14, we present three frame-based solutions, where *Seg* means splitting along each GOP. On the right side, we present five methods based on tiles, with *Column* indicating the use of the column-wise parallel decoding method (Section 4.2). When comparing *1(Frame)* and *8(seg)*, we observed that the root cause of acceleration achieved by TVM is tiling. Furthermore, the decoding time decreases by 33.8% when increasing the threads from 1 to 8, but stabilizes as the thread count further increases. Therefore, the default implementation of TVM uses 8 threads. Consequently, our method avoids about 36% of computational waste compared with the *Column* schema, as it involves decoding tiles that are not required, highlighting the effectiveness of our row-wise parallel decoding method.

Figure 15 illustrates the impact of gradually incorporating cache algorithms when executing the *track* query, where *w/o cache* represents the scenario where all cache algorithms are disabled. The experimental results demonstrate that adding the codec caching technology reduces the decoding overhead by 3%. Subsequently, the inclusion of the state caching and tile caching algorithms further saves 3% and 8%, respectively. Generally, the proposed caching algorithms effectively avoid computational waste and save more than 14% of decoding time in total. Notably, the tile caching strategy plays a predominant role in achieving these savings. This is because the cached decoders are preemptive resources facing contention in a multi-threaded environment. In this case, the acceleration effect of parallel decoding is weakened. As the state caching algorithm is built upon the codec caching algorithm, it is also influenced by contention issues. In contrast, since multiple threads do not access the same tile simultaneously, the tile caching strategy avoids conflict problems. Moreover, with the assistance of tile caching, multiple tiles can be retrieved without the involvement of the decoder, which effectively mitigates decoder contention.

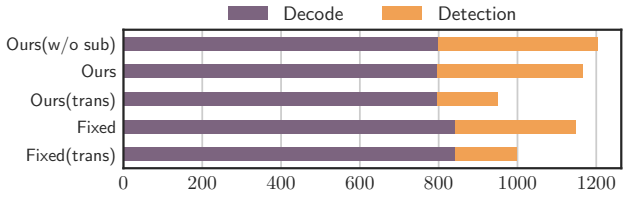


Figure 16: Sub-layouts improve inference performance (s).

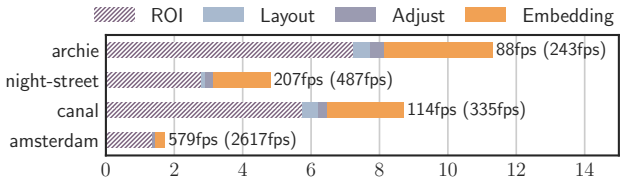


Figure 17: Indexing overhead (ms).

5.7 Effectiveness of Hierarchical Layout

We conducted *track* queries on the archie dataset with different configurations to demonstrate the effectiveness of the hierarchical layout. *Ours(w/o sub)* denotes the scenario where the sub-layout is discarded, and the full tile is inputted into the object detection DNN. *Fixed* refers to a configuration where both random access cost $F_{\mathscr{L}}$ and inferring overhead $F^{nn}_{\mathscr{L}}$ are considered during tile storage. The default DNN used in this experiment is YoloV5m (medium capacity). Additionally, *Ours(trans.)* and *Fixed(trans.)* represent the usage of YoloV5s (small capacity), a lighter but less accurate DNN, for querying on the layout generated for YoloV5m. By employing the sub-layout, TVM achieves an additional 9% reduction in the inference time of the object detection DNN. The *Fixed* schema, which considers both types of overhead, demonstrates optimal overall performance on the pre-determined DNN. However, it does not adapt well to the transfer of DNN models. In contrast, *Ours(trans)* adjusts the sub-layout before inference, resulting in improved performance in this scenario. This shows that TVM can switch between different DNNs more flexibly. It is worth noting that, given a video stream, *e.g.*, a HEVC [48] steam, different decoder implementations [9, 39] yield identical decoding results. However, the processing results of DNNs may vary significantly. Therefore, the adaptability of the framework to the DNNs is far more crucial than that to the decoders.

5.8 Index Establishment Efficiency

TVM constructs the index during the data ingestion process, encompassing four primary stages: ROI extraction, layout generation, layout adjustment, and semantic embedding generation. Since the background subtraction algorithms can be executed on the smart camera, we show the throughput with (w/o) ROI generation in Figure 17. It indicates that TVM effectively achieves real-time index construction across all datasets. Apart from video resolution, the intricacy of content stands as another pivotal determinant influencing the indexing process. Videos characterized by intricate content tend to yield a greater number of smaller ROIs, consequently demanding more time for TVM to ingest them. For instance, the water wave movements in the canal dataset often lead to many ROIs.

6 RELATED WORK

Early works in visual similarity search [4, 5, 34, 46, 58] leverage hand-crafted features to extract semantic content, contributing valuable insights to the management of video data before the advent of deep learning techniques. Query frameworks often aim to minimize the calls to computationally expensive DNNs. NoScope [25], BlazeIt [24] and SUPG [26] use lightweight specialized NNs, while MIRIS [7] and ExSample [37] use NN-guided and Thompson sampling, respectively. FiGO [15] segments videos into *chunks* and selects the most suitable model for each chunk. VIVA [43] uses relational hints to optimize complex video analytic queries. OTIF [8] efficiently extracts all tracks at the pre-processing stage. TVM can contribute to these frameworks by enabling faster retrieval of video frames. TASTI [27] generates frame-based semantic embedding, and the *Frame* scheme in our experiments is an implementation of it. Optasia [35], Scanner [40], LightDB [19], and VideoEdge [22] leverage query parallelism or distributed computing, which can be combined with TVM to achieve higher efficiency. Storage frameworks optimize storage and retrieval performance. VStore [54] accelerates the entire query process by creating multiple copies of the video with different encoding configurations. VSS [18] employs caching techniques to store frequently retrieved regions in various formats. Smol [28] optimizes the resolution of visual data and architecture or NNs to achieve trade-offs between accuracy and throughput. These frameworks accelerate video retrieval by caching copies in different formats, which can be integrated with TVM. TASM [16] manages and retrieves videos in tiles, which focuses on optimizing the sub-frame selection. Our layout adjustment algorithms can further enhance its performance. Consequently, some methods [6, 32, 33, 45] analyze video in the compressed domain, which can be used to generate the ROI required by TVM.

7 CONCLUSION

In this work, we present TVM, a novel tile-based video management framework that optimizes the decoding and detection processes in video analytic queries. Specifically, TVM constructs a tile-based semantic index, which significantly accelerates various queries by reducing the decoding and processing of irrelevant pixels. Meanwhile, we propose a query-driven tile parallel decoding algorithm and resource caching algorithms to effectively avoid unnecessary computation and minimize repeated calculations, respectively. To demonstrate the effectiveness of TVM, we evaluate it on various large-scale datasets and query methods, resulting in a speedup of more than 5.6× compared to the frame-based methods.

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Program of China (Nos. 2020YFB1707900, 2021YFB2700700), National Natural Science Foundation of China (Nos. U23B2019, U23A20297, 62072035, 62102024). Ye Yuan is supported by the National Key R&D Program of China (No. 2022YFB2702100), the NSFC (Nos. 61932004, 62225203, U21A20516) and the DITDP (No. JCKY2021211B017).

REFERENCES

[1] Michael R Anderson, Michael Cafarella, German Ros, and Thomas F Wenisch. 2019. Physical representation-based predicate optimization for a visual analytics database. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1466–1477.

[2] Kittipat Apicharttrisorn, Xukan Ran, Jiasi Chen, Srikanth V Krishnamurthy, and Amit K Roy-Chowdhury. 2019. Frugal following: Power thrifty object detection and tracking for mobile augmented reality. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. 96–109.

[3] Olivier Barnich and Marc Van Droogenbroeck. 2010. ViBe: A universal background subtraction algorithm for video sequences. *IEEE Transactions on Image processing* 20, 6 (2010), 1709–1724.

[4] Ilaria Bartolini, Paolo Ciaccia, Lei Chen, Vincent Oria, et al. 2006. A meta-index to integrate specific indexes: application to multimedia. In *12th International Conference on Distributed Multimedia Systems (DMS 2006)*. Citeseer, 29–36.

[5] Ilaria Bartolini, Marco Patella, and Corrado Romani. 2013. SHIATSU: tagging and retrieving videos without worries. *Multimedia tools and applications* 63 (2013), 357–385.

[6] Madhushree Basavarajaiah and Priyanka Sharma. 2019. Survey of compressed domain video summarization techniques. *ACM Computing Surveys (CSUR)* 52, 6 (2019), 1–29.

[7] Favyen Bastani, Songtao He, Arjun Balasingam, Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan, Michael Cafarella, Tim Kraska, and Sam Madden. 2020. Miris: Fast object track queries in video. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1907–1921.

[8] Favyen Bastani and Samuel Madden. 2022. OTIF: Efficient tracker pre-processing over large video datasets. In *Proceedings of the 2022 International Conference on Management of Data*. 2091–2104.

[9] Bellard. 2018. FFmpeg. https://ffmpeg.org. Accessed: 2023-05-11.

[10] Yannick Benezeth, Pierre-Marc Jodoin, Bruno Emile, Hélène Laurent, and Christophe Rosenberger. 2008. Review and evaluation of commonly-implemented background subtraction algorithms. In *2008 19th International Conference on Pattern Recognition*. IEEE, 1–4.

[11] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. 2016. Simple online and realtime tracking. In *2016 IEEE international conference on image processing (ICIP)*. IEEE, 3464–3468.

[12] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934* (2020).

[13] Thierry Bouwmans, Fida El Baf, and Bertrand Vachon. 2008. Background modeling using mixture of gaussians for foreground detection-a survey. *Recent patents on computer science* 1, 3 (2008), 219–237.

[14] Benjamin Bross, Ye-Kui Wang, Yan Ye, Shan Liu, Jianle Chen, Gary J Sullivan, and Jens-Rainer Ohm. 2021. Overview of the versatile video coding (VVC) standard and its applications. *IEEE Transactions on Circuits and Systems for Video Technology* (2021).

[15] Jiashen Cao, Karan Sarkar, Ramyad Hadidi, Joy Arulraj, and Hyesoon Kim. 2022. FiGO: Fine-Grained Query Optimization in Video Analytics. In *Proceedings of the 2022 International Conference on Management of Data*. 559–572.

[16] Maureen Daum, Brandon Haynes, Dong He, Amrita Mazumdar, and Magdalena Balazinska. 2021. TASM: A tile-based storage manager for video analytics. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 1775–1786.

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[18] Brandon Haynes, Maureen Daum, Dong He, Amrita Mazumdar, Magdalena Balazinska, Alvin Cheung, and Luis Ceze. 2021. Vss: A storage system for video analytics. In *Proceedings of the 2021 International Conference on Management of Data*. 685–696.

[19] Brandon Haynes, Amrita Mazumdar, Magdalena Balazinska, Luis Ceze, and Alvin Cheung. 2018. Lightdb: A dbms for virtual reality video. *Proceedings of the VLDB Endowment* 11, 10 (2018).

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[21] Marko Heikkila and Matti Pietikainen. 2006. A texture-based method for modeling the background and detecting moving objects. *IEEE transactions on pattern analysis and machine intelligence* 28, 4 (2006), 657–662.

[22] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. 2018. Videoedge: Processing camera streams using hierarchical clusters. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 115–131.

[23] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, Yonghye Kwon, Kalen Michael, Jiacong Fang, Zeng Yifu, Colin Wong, Diego Montes, et al. 2022. ultralytics/yolov5: v7. 0-YOLOv5 SOTA Realtime Instance Segmentation. *Zenodo* (2022).

[24] Daniel Kang, Peter Bailis, and Matei Zaharia. 2018. BlazeIt: optimizing declarative aggregation and limit queries for neural network-based video analytics. *arXiv preprint arXiv:1805.01046* (2018).

[25] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. Noscope: optimizing neural network queries over video at scale. *arXiv preprint arXiv:1703.02529* (2017).

[26] Daniel Kang, Edward Gan, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2020. Approximate selection with guarantees using proxies. *arXiv preprint arXiv:2004.00827* (2020).

[27] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2020. Semantic Indexes for Machine Learning-based Queries over Unstructured Data. *arXiv e-prints* (2020), arXiv–2009.

[28] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia. 2020. Jointly optimizing preprocessing and inference for DNN-based visual analytics. *arXiv preprint arXiv:2007.13005* (2020).

[29] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. 2015. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 1–6.

[30] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. 2020. 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*. 359–376.

[31] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision*. 2980–2988.

[32] Huan Liu, Wentao Liu, Zhixiang Chi, Yang Wang, Yuanhao Yu, Jun Chen, and Jin Tang. 2022. Fast Human Pose Estimation in Compressed Videos. *IEEE Transactions on Multimedia* 25 (2022), 1390–1400.

[33] Qiankun Liu, Bin Liu, Yue Wu, Weihai Li, and Nenghai Yu. 2022. Real-time online multi-object tracking in compressed domain. *arXiv preprint arXiv:2204.02081* (2022).

[34] Yan Liu and Fei Li. 2002. Semantic extraction and semantics-based annotation and retrieval for video databases. *Multimedia Tools and Applications* 17 (2002), 5–20.

[35] Yao Lu, Aakanksha Chowdhery, and Srikanth Kandula. 2016. Optasia: A relational platform for efficient large-scale video analytics. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*. 57–70.

[36] Antoine Manzanera and Julien C Richefeu. 2007. A new motion detection algorithm based on Σ–Δ background estimation. *Pattern Recognition Letters* 28, 3 (2007), 320–328.

[37] Oscar Moll, Favyen Bastani, Sam Madden, Mike Stonebraker, Vijay Gadepally, and Tim Kraska. 2022. Exsample: Efficient searches on video repositories through adaptive sampling. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2956–2968.

[38] Nvidia. 2013. Nvidia Video Codec. https://developer.nvidia.com/video-codec-sdk. Accessed: 2023-05-11.

[39] Nvidia. 2019. Video Processing Framework. https://github.com/NVIDIA/VideoProcessingFramework. Accessed: 2023-05-11.

[40] Alex Poms, Will Crichton, Pat Hanrahan, and Kayvon Fatahalian. 2018. Scanner: Efficient video analysis at scale. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–13.

[41] Joseph Redmon and Ali Farhadi. 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).

[42] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*. 91–99.

[43] Francisco Romero, Johann Hauswald, Aditi Partap, Daniel Kang, Matei Zaharia, and Christos Kozyrakis. 2022. Optimizing Video Analytics with Declarative Model Relationships. *Proceedings of the VLDB Endowment* 16, 3 (2022), 447–460.

[44] Heiko Schwarz, Detlev Marpe, and Thomas Wiegand. 2007. Overview of the scalable video coding extension of the H. 264/AVC standard. *IEEE Transactions on circuits and systems for video technology* 17, 9 (2007), 1103–1120.

[45] Sandeep Singh Sengar and Susanta Mukhopadhyay. 2020. Moving object detection using statistical background subtraction in wavelet compressed domain. *Multimedia Tools and Applications* 79, 9-10 (2020), 5919–5940.

[46] Arnold WM Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. 2000. Content-based image retrieval at the end of the early years. *IEEE Transactions on pattern analysis and machine intelligence* 22, 12 (2000), 1349–1380.

[47] Pierre-Luc St-Charles and Guillaume-Alexandre Bilodeau. 2014. Improving background subtraction using local binary similarity patterns. In *IEEE winter conference on applications of computer vision*. IEEE, 509–515.

[48] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, Thomas Wiegand, et al. 2012. Overview of the high efficiency video coding(HEVC) standard. *TCSVT* 22, 12 (2012), 1649–1668.

[49] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. 2019. Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF International Conference*

on Computer Vision. 9627–9636.

[50] Junjue Wang, Ziqiang Feng, Zhuo Chen, Shilpa George, Mihir Bala, Padmanabhan Pillai, Shao-Wen Yang, and Mahadev Satyanarayanan. 2018. Bandwidth-efficient live video analytics for drones via edge computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 159–173.

[51] Xiaoli Wang, Aakanksha Chowdhery, and Mung Chiang. 2016. SkyEyes: adaptive video streaming from UAVs. In *Proceedings of the 3rd Workshop on Hot Topics in Wireless*. 2–6.

[52] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. 2017. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*. IEEE, 3645–3649.

[53] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1492–1500.

[54] Tiantu Xu, Luis Materon Botelho, and Felix Xiaozhu Lin. 2019. Vstore: A data store for analytics on large videos. In *Proceedings of the Fourteenth EuroSys Conference 2019*. 1–17.

[55] Jian Yao and Jean-Marc Odobez. 2007. Multi-layer background subtraction based on color and texture. In *2007 IEEE conference on computer vision and pattern recognition*. IEEE, 1–8.

[56] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live video analytics at scale with approximation and delay-tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation*.

[57] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Haibin Lin, Zhi Zhang, Yue Sun, Tong He, Jonas Mueller, R Manmatha, et al. 2020. Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955* (2020).

[58] Lei Zhang and Yong Rui. 2013. Image search—from thousands to billions in 20 years. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 9, 1s (2013), 1–20.

[59] Zoran Zivkovic. 2004. Improved adaptive Gaussian mixture model for background subtraction. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, Vol. 2. IEEE, 28–31.

[60] Zoran Zivkovic and Ferdinand Van Der Heijden. 2006. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern recognition letters* 27, 7 (2006), 773–780.