2021 IEEE 37th International Conference on Data Engineering (ICDE) | 978-1-7281-9184-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICDE51399.2021.00156

TASM: A Tile-Based Storage Manager for Video Analytics

Maureen Daum*, Brandon Haynes†, Dong He*, Amrita Mazumdar*, Magdalena Balazinska*
*Paul G. Allen School of Computer Science & Engineering, University of Washington
{mdaum, donghe, amrita, magda}@cs.washington.edu
†Gray Systems Lab, Microsoft
Brandon.Haynes@microsoft.com

***Abstract*—Modern video data management systems store videos as a single encoded file, which significantly limits possible storage level optimizations. We design, implement, and evaluate TASM, a new tile-based storage manager for video data. TASM uses a feature in modern video codecs called "tiles" that enables spatial random access into encoded videos. TASM physically tunes stored videos by optimizing their tile layouts given the video content and a query workload. Additionally, TASM dynamically tunes that layout in response to changes in the query workload or if the query workload and video contents are incrementally discovered. Finally, TASM also produces efficient initial tile layouts for newly ingested videos. We demonstrate that TASM can speed up subframe selection queries by an average of over 50% and up to 94%. TASM can also improve the throughput of the full scan phase of object detection queries by up to 2×.**

I. Introduction

The proliferation of inexpensive high-quality cameras coupled with recent advances in machine learning and computer vision have enabled new applications on video data such as automatic traffic analysis [1], [2], retail store planning [3], and drone analytics [4], [5]. This has led to a class of database systems specializing in video data management that facilitate query processing over videos [3], [6]–[10].

A query over a video comprises two steps. First, read the video file from disk and decode it. Second, process frames to identify and return pixels of interest or compute an aggregate. Most systems, so far, have focused on accelerating and optimizing the second step [3], [9]–[11], often assuming that the video is already decoded and stored in memory [3], [7], [12], which is not feasible in practice.

The lack of efficient storage managers in existing video data management systems significantly impacts queries. First, *subframe selection* queries (e.g., "Show me video snippets cropped to show previously identified hummingbirds feeding on honeysuckles") are common and their execution bottleneck is at the storage layer since these queries are selections, reading and returning pixels without additional operations. Second, *object detection* queries, which extract new semantic information from a video (e.g., "Find all sightings of hummingbirds in this new video") require the execution of expensive deep learning models. To avoid applying such models to as many frames as possible, query plans typically include an initial *full scan* phase that applies a cheap predicate [12] or a specialized model [7] to the entire video to filter uninteresting frames. The overhead of reading and decoding the video file is known to significantly hurt the performance of this phase [13].

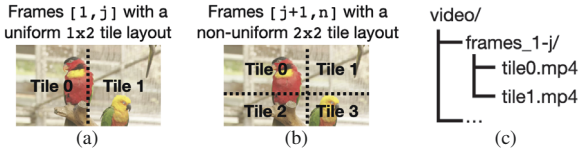


Fig. 1. Video partitioned into tiles. (a) shows the first j frames partitioned with a uniform 1×2 layout. (b) shows frames partitioned with a non-uniform 2×2 layout. (c) shows a directory hierarchy. Video stored at `video/frames_1-j/tile0.mp4` contains the left half of frames $[1, j]$.

In this paper, we introduce TASM, a storage manager that greatly improves the performance of subframe selection queries and the full scan phase of object detection queries by providing spatial random access within videos. TASM exploits the observation that objects in videos frequently lie in subregions of video frames. For example, a traffic camera may be oriented such that it partially captures the sky, so vehicles only appear in the lower portion of a frame. Analysis applications such as running license plate recognition [1] or extracting image patches for vehicle type recognition [1] only need to operate on the parts of the frame containing vehicles. Privacy applications such as blurring license plates and faces [14] or performing region of interest-based encryption [15] similarly only need to modify the parts of the frame that contain sensitive objects.

Using its spatial random access capability, TASM enables reading from disk and decoding only the parts of the frame that are interesting to queries. Providing such a capability is difficult because the video encoding process introduces spatial and temporal dependencies within and between frames. To address this problem, TASM subdivides video frames into smaller pieces called *tiles* that can be processed independently. As shown in Fig. 1, each tile contains a rectangular subregion of the frame that can be decoded independently because there are no spatial dependencies between tiles. In contrast, current state of the art incurs the cost of decoding entire frames. TASM optimizes how a video is divided into tiles and stored on disk to reduce the amount of work spent decoding and preprocessing parts of the video not involved in a query. Through its use of tiles, TASM implements a new type of optimization that we call *semantic predicate pushdown* where predicates are pushed below the decoding step and only tiles of interest are read from

978-1-7281-9184-3/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDE51399.2021.00156

disk, decoded, and processed.

Building TASM raises three challenges. The first challenge is fundamental, but important: Given a video file with known semantic content (i.e., known object locations within video frames) and a known query workload, TASM must decide on the optimal tile layout, choosing from among layouts with uniform or non-uniform tiles and either fine-grained or coarse-grained tiles. TASM must also decide whether different tile layouts should be used in different parts of a video. To do this effectively, TASM must accurately estimate the cost of executing a query with a given tile layout. TASM therefore drives its selection using a cost function that balances the benefits of processing fewer pixels against the overhead of processing more tiles for a given tile layout, video content, and query workload. In this paper, we experimentally demonstrate that non-uniform, fine-grained tiles outperform the other options. Additionally, we find that optimizing the layout for short sections of the video (i.e., every 1 second) maximizes query performance with no storage overhead. Given a video file, TASM thus splits it into 1 second fragments and selects the optimal fine-grained tile layout for each fragment.

The second challenge is that the semantic content and the query workload for a video are typically discovered over time as users execute object detection and subframe selection queries. TASM therefore lacks the information it needs to design optimal tile layouts. To address this challenge, TASM incrementally updates a video's tile layout as queries to detect and retrieve objects are executed. TASM uses different tile layouts in different parts of the video, and independently evolves the tile layout in each section. To do this, TASM builds on techniques from database cracking [16], [17] and online indexing [18]. To decide when to re-tile portions of the video and which layout to use, TASM maintains a limited set of alternative layouts based on past queries. It then uses its cost function to accumulate estimated performance improvements offered by these tile layouts as it observes queries. Once the estimated improvement, also called regret [19], of a new layout offsets the cost of reorganization, TASM re-tiles that portion of the video. By observing multiple queries before making tiling decisions, TASM designs layouts optimized for multiple query types. For the ornithology example, TASM could tile around hummingbirds *and* flowers that are likely to attract them.

The third challenge lies in the initial phase that identifies objects of interest in a new video. This phase is both expensive and requires at least one full scan over the video, generally using a cheap model to filter frames or compute statistics. The models used in the full scan phase are limited by video decoding and preprocessing throughput [13]. To address this final challenge, TASM uses semantic predicate pushdown where the semantic predicate is not a specific object type, but rather a general region of interest (ROI). TASM bootstraps an initial tile layout using an inexpensive predicate that identifies ROIs within frames. This predicate can use background segmentation to find foreground objects, motion vectors to identify areas with large amounts of motion, or even a specialized neural network designed to identify specific object types. When an object detection query is executed, TASM only decodes the tiles that contain ROIs, hence filtering regions of the frame before the decode step. TASM thus alleviates the bottleneck for the full scan phase of object detection queries by reducing the amount of data that must be decoded and preprocessed. TASM can be directly incorporated into existing techniques and systems that accelerate the extraction of semantic information from videos (e.g., [3], [11]).

In summary, the contributions of this paper are as follows:

- We develop TASM[1], a new type of storage manager for video data that splits video frames into independently queryable tiles. TASM optimizes the tile layout of a video file based on its contents and the query workload. By doing so, TASM accelerates queries that retrieve objects in videos while keeping storage overheads low and maintaining good video quality.
- We develop new algorithms for TASM to dynamically evolve the video layout as information about the video content and query workload becomes available over time.
- We extend TASM to cheaply profile videos and design an initial layout around ROIs when a video is initially ingested. This initial tiling reduces the preprocessing work required for object detection queries.

We evaluate TASM on a variety of videos and workloads and find that the layouts picked by TASM speed up subframe selection queries by an average of 51% and up to 94% while maintaining good quality, and that TASM automatically tunes layouts after just a small number of queries to improve performance even when the workload is unknown. We also find that TASM improves the throughput of the full scan phase of object detection by up to 2× while maintaining high accuracy.

II. Background

Videos are stored as encoded files due to their large size. Video codecs such as H264 [20], HEVC [21], and AV1 [22] specify algorithms used to (de)compress videos. While the specific algorithms used by various codecs differ, the high-level approach is the same as we describe in this section.

Groups of pictures: A video consists of a sequence of frames, where each frame is a 2D array of pixels. Frames in the sequence are partitioned into *groups of pictures* (GOPs). Each GOP is encoded independently from the other GOPs and is typically one second in duration. The first frame in a GOP is called a *keyframe*. Keyframes allow GOPs to act as temporal random access points into the video because it is possible to start decoding a video at any keyframe. To retrieve a specific frame, the decoder begins decoding at the closest keyframe preceding the frame being retrieved. Keyframes have large storage sizes because they use a less efficient form of compression than other types of frames, so the number of keyframes impacts a video's overall storage size. Videos stored with long GOPs are smaller in size than videos stored with short GOPs, but they also have fewer random access opportunities.

[1]Code is available at https://github.com/uwdb/TASM.

Tiles: Compressed videos do not generally support decoding spatial regions of a frame. The encoding process creates spatial dependencies within a frame, and decoders must resolve these dependencies by decoding the entire frame, even if just a small region is requested. Modern codecs, however, provide a feature called *tiles* that enables splitting frames into independently-decodable regions. Fig. 1 illustrates this concept. Like frames, tiles are also 2D arrays of pixels. However, a tile only contains the pixels for a rectangular portion of the frame. The full frame is recovered by combining the tiles. Tiles introduce spatial random access points for decoding. To decode a region within a frame, only the tiles that contain the requested region are processed. This flexibility to decode spatial subsets of frames comes with tradeoffs in quality; tiling can lead to artifacts appearing at the tile boundaries [23], which reduces the visual quality of videos. As such, carefully selecting tile layouts is important for high-quality query results. While tiles act as spatial random access points, temporal random access is still provided by keyframes. Tiles are applied to all frames within a GOP, so decoding a tile in a non-keyframe requires decoding that tile in all frames starting from the preceding keyframe.

A *tile layout* defines how a sequence of frames is divided into tiles. A layout $L{=}(n_r, n_c, \{h_1, \ldots, h_{n_r}\}, \{w_1, \ldots, w_{n_c}\})$ is defined by the number of rows and columns, n_r and n_c, the height of each row, and the width of each column. These parameters define the (x, y) offset, width, and height of the $n_r{\cdot}n_c$ tiles. An untiled video is a special case of a tile layout consisting of a single tile that encompasses the entire frame: $\omega = (1, 1, \{frame\_height\}, \{frame\_width\})$. Valid layouts require tiles to be partitioned along a regular grid, meaning rows and columns extend through the entire frame. We do not consider irregular layouts, which are not supported by the HEVC specification [21]. Different tile layouts can be used throughout the video; a *sequence of tiles* (SOT) refers to a sequence of frames with the same tile layout. Changes in the tile layout must happen at GOP boundaries, so every new layout must start at a keyframe. Therefore, changing the tile layout has a high storage overhead for the same reason that starting a new GOP has a high storage overhead. The cost of executing a query over a video encoded with tiles is proportional to the number of pixels and tiles that are decoded.

Stitching: Tiles can be stored separately, but they must be combined to recover the original video. Tiles can be combined without an intermediate decode step using a process called *homomorphic stitching* [24]. Homomorphic stitching interleaves the encoded data from each tile and adds header information so the decoder knows how the tiles are arranged.

III. Tile-based storage manager design

In this section, we present the design of TASM, our tile-based storage manager. TASM is designed to be the lowest layer in a VDBMS. Unlike existing storage managers that serve requests for sequences of frames, TASM efficiently retrieves regions *within* frames to answer queries for specific objects.

Fig. 2 shows an overview of how TASM integrates with the rest of a VDBMS. TASM incrementally populates a *semantic index* to store the bounding boxes associated with object detections. While queries for statistics about the semantic content can use the semantic index to avoid re-running expensive analysis over the frame contents, TASM uses this index to generate tile layouts, split videos into tiles, store such physically tuned videos as files, and answer content-based queries more efficiently by retrieving only relevant tiles from disk.

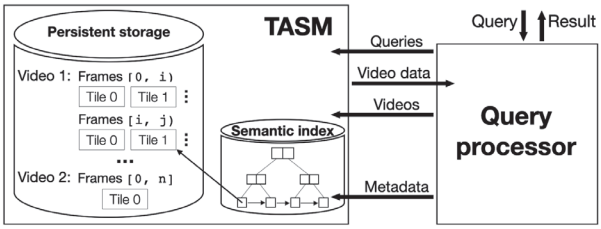


Fig. 2. Overview of how TASM integrates with a VDBMS.

A. TASM API

TASM exposes the following access method API:

Method	Parameters	Result
Scan	$video$, $L : labels$, $T : times$	Pixel[]
AddMetadata	$video$, $frame$, $label$, x_1, y_1, x_2, y_2	—

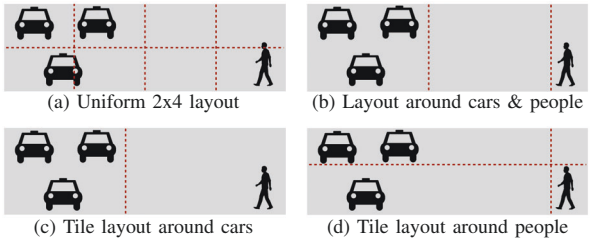
The core method Scan $(video, L, T)$ performs subframe selection by retrieving the pixels that satisfy a CNF predicate on the labels, L, and an optional predicate on the time dimension, T. For example, $L{=}(label{=}\text{`}car\text{'}){\vee}(label{=}\text{`}bicycle\text{'})$ retrieves pixels for both cars and bicycles.

TASM also exposes an API to incorporate metadata generated during query processing into the semantic index (discussed in the following section). The method AddMetadata $(video, frame, label, x_1, y_1, x_2, y_2)$ adds the bounding box (x_1, y_1, x_2, y_2) on $frame$ to the semantic index and associates it with the specified label.

B. Semantic index

TASM maintains metadata about the contents of videos in a *semantic index*. The semantic information consists of labels associated with bounding boxes. Labels denote object types and properties such as color. Bounding boxes locate an object within a frame. When the query processor invokes TASM's Scan method, TASM must efficiently retrieve bounding box information associated with the specified parameters. The semantic index is therefore implemented as a B-tree clustered on $(video, label, time)$. The leaves contain information about the bounding boxes and pointers to the encoded video tile(s) each box intersects based on the associated tile layout.

The semantic index is populated through the AddMetadata method as object detection queries execute. As we discuss in Section IV, TASM creates an initial layout around high-level regions of interest within frames to speed up object detection queries. As those queries execute and add more objects to the semantic index, TASM incrementally updates the tile layout to maximize the performance of the observed query workload.



(a) Uniform 2x4 layout (b) Layout around cars & people

(c) Tile layout around cars (d) Tile layout around people

Fig. 3. Various ways to tile a frame. (a) is a uniform layout, while (b)-(d) are non-uniform layouts. Depending on which objects are targeted, different layouts will be more efficient.

(a) Fine-grained tiles (b) Coarse-grained tiles

Fig. 4. Non-uniform tile layout around cars using (a) fine-grained tiles, or (b) coarse-grained tiles.

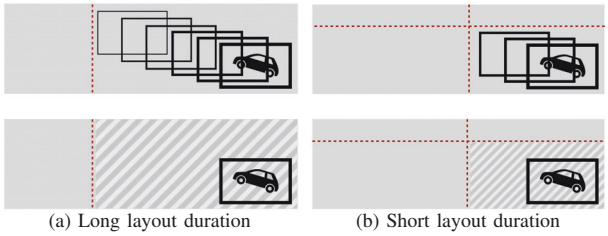
C. Tile-based data storage

Having captured the metadata about objects and other interesting areas in a video using the semantic index, the next step is to leverage it to guide how the video data is encoded with tiles. Two tiling approaches are possible: uniform-sized tiles, or non-uniform tiles whose dimensions are set based on the locations of objects in the video. Both techniques can improve query performance, but tile layouts that are designed around the objects in frames can reduce the number of non-object pixels that have to be decoded. Fig. 3 shows these different tiling strategies on an example frame.

1) Uniform layouts: The uniform layout approach divides frames into tiles with equal dimensions. This approach does not leverage the semantic index, but if objects in the video are small relative to the total frame size, they will likely lie in a subset of the tiles. However, an object can intersect multiple tiles, as shown in Fig. 3a where part of the person lies in two tiles. While TASM decodes fewer pixels than the entire frame, it still must process many pixels that are not requested by the query. Further, the visual quality of the video is reduced because in general a large number of uniform tiles are required to improve query performance, as shown in Fig. 7b.

2) Non-uniform layouts: TASM creates non-uniform layouts with tile dimensions such that objects targeted by queries lie within a single tile. There are a number of ways a given tile layout can benefit multiple types of queries. If a large portion of the frame does not contain objects of interest, the layout can be designed such that this region does not have to be processed. If objects of interest appear near each other, a single tile around this region benefits queries for any of these objects. If objects are not nearby but do appear in clusters, creating a tile around each cluster can also accelerate queries for these objects.

Fig. 4 shows examples of non-uniform layouts around cars. For a set of bounding boxes B, TASM picks tile boundaries guided by a desired tile granularity. For coarse-grained tiles (Fig. 4b), it places all B within a single, large tile. For fine-grained tiles (Fig. 4a), it attempts to isolate non-intersecting $b \in B$ into smaller tiles while respecting minimum tile dimensions specified by the codec and ensuring that no tile boundary intersects any $b \in B$. TASM does not limit the number of tiles in a layout. To modulate quality, this could be made a user-specified setting; we leave this as future work. TASM processes fewer pixels from a video stored with fine-grained tiles because the tiles do not contain the parts of the frame between objects, but it processes more individual tiles because multiple tiles in each frame may contain objects. TASM estimates the overall effectiveness of a layout using a cost function that combines these two metrics, as described in Section IV-A.



(a) Long layout duration (b) Short layout duration

Fig. 5. (a) shows how more pixels must be decoded on each individual frame when a tile layout extends for many frames compared to (b) where fewer frames have the same layout. The boxes show the location of the car on later frames, and the dashed lines show the tile boundaries. The striped region indicates the tile that would be decoded for a query targeting cars.

In addition to deciding the tile granularity, TASM also chooses *which* objects to design the tile layout around, and therefore which bounding boxes to include in B. The best choice depends on the queries. For example, if queries target people, a layout around just people, as in Fig. 3d, is more efficient than a layout around both cars and people (Fig. 3b). We explain how TASM makes this choice in Section IV.

3) Temporally-changing layouts: Different tile layouts, uniform and non-uniform, can be used throughout a video; the layout can change as often as every GOP. TASM uses different layouts throughout a video to adapt to objects as they move.

The size of these temporal sections is determined by the *layout duration*, which refers to the number of frames within a sequence of tiles (SOT). Layout duration is separate from GOP length; while layout duration cannot be shorter than a GOP, it can extend over multiple GOPs. The layout duration affects the sizes of tiles in non-uniform layouts, as shown in Fig. 5. In general, longer tile layout durations have lower storage costs but lead to larger tiles because TASM must consider more object bounding boxes as objects move and new objects appear. Therefore, TASM must decode more data on each frame. We evaluate this tradeoff in Fig. 10.

4) Not tiling: Layouts that require TASM to decode a similar number of pixels as when the video is not tiled can actually slow queries down due to the implementation complexities that arise from working with multiple tiles. Therefore, TASM may opt to not tile GOPs when the gain in performance does not exceed a threshold value.

5) Data storage and retrieval: TASM stores each tile as a separate video file, as shown in Fig. 1. If different layouts are used throughout the video, each tile video contains only the frames with that layout. If only a segment of a video is ever queried, TASM reads and tiles just the frames in that segment.

This storage structure facilitates the ingestion of new videos because each video's data is stored separately. Additionally, because each GOP is also stored separately, to modify an existing video, updated GOPs can replace original ones, or new GOPs can be appended.

TASM retrieves just the tiles containing the objects targeted by queries. When complete frames are requested, TASM applies homomorphic stitching (see Section II). This stitching process can also be used to efficiently convert the tiles into a codec-compliant video that other applications can interact with.

IV. Tiling strategies

TASM automatically tunes the tile layout of a video to improve query performance. The objects in a video and workloads, or the set of queries presented to a VDBMS, may be *known* or *unknown*. When TASM has full knowledge of both the objects targeted by queries and the locations of these objects in video frames, TASM designs tile layouts before queries are processed, as described in Section IV-B. In practice, the objects targeted by queries and their locations are initially unknown. TASM uses techniques from online indexing to incrementally design layouts based on prior queries and the objects detected so far, as described in Section IV-C. Finally, TASM also creates an efficient, initial tiling before any queries are executed as we present in Section IV-D.

A. Notation and cost function

We first introduce notation that will be used throughout this section. A query workload $Q = (q_1, ..., q_n)$ is a list of queries, where each query requests pixels belonging to specified object classes, possibly with temporal constraints. The set O_{q_i} represents the objects requested by an individual query q_i, while $O_Q = \cup_{q_i \in Q} O_{q_i}$ is the set of all objects targeted by Q.

A video $v = s_0 \oplus \cdots \oplus s_n$ is a series of concatenated, non-overlapping, non-empty sequence of tiles (SOTs; see Section II), s_i. A video layout specification $\mathscr{L}{=}s_i \mapsto L$ maps each SOT to a tile layout, L, which specifies how frames are partitioned into tiles, as described in Section II. If a SOT is not tiled, then $s_i{\mapsto}\omega$, where ω refers to a $1{\times}1$ tile layout. $\textsc{partition}(s, O)$ refers to tiling the SOT using a non-uniform layout around the bounding boxes associated with objects in the set O using the techniques from Section III-C2. For example, $\textsc{partition}(s, \{car, person\})$ refers to creating a layout around cars and people, as in Fig. 3b.

TASM implements a "what-if" interface [25] to estimate the cost of executing queries with alternative layouts using a cost function. The estimated cost of executing query q over SOT s encoded with layout L is $C(s, q, L){=}\beta \cdot P(s, q, L) + \gamma \cdot T(s, q, L)$. The cost C is proportional to the number of pixels P, and the number of tiles T that are decoded, both of which depend on the query and layout. To validate this cost function and estimate β and γ to use in experiments, we fit a linear model to the decode times for over $1{,}400$ video, query, and non-uniform layout combinations used in the microbenchmarks in Section V-B. The resulting model achieves $R^2{=}0.996$. The exact values of β and γ will depend on the system; TASM can re-estimate them by generating a number of layouts from a small sample of videos and measuring execution time.

Finally, the cost of executing q over video v encoded with layout specification $\mathscr{L}$is the sum of its SOT costs (i.e., $C(v, q, \mathscr{L}){=}\sum_{s_i \in v} C(s_i, q, \mathscr{L}(s_i))$) and the cost of executing an entire query workload is the sum over all individual queries, $C(v, Q, \mathscr{L}){=}\sum_{q_i \in Q} C(v, q_i, \mathscr{L})$. The difference in estimated query time for query q over SOT s between layouts L and L' is $\Delta(q, L, L', s){=}C(s, q, L){-}C(s, q, L')$, or simply $\Delta(q, L, L')$ when s is obvious from the context. The cost of (re-)encoding SOT s with layout L is $R(s, L)$.

Using this cost function, the maximum expected improvement for an individual query is inversely proportional to the object density, which determines the number of pixels (P) and tiles (T). Tiling therefore leads to negligible improvement—or even regressions—when objects are dense and occupy a large fraction of a frame. In those cases, TASM does not tile a video at all as we discuss in Section IV-B. In contrast, tiling yields large improvements when objects are sparse. Fig. 11 shows the linear relationship. It shows how, for a given video and query, non-uniform tiling reduces the number of pixels that must be decoded, which directly increases performance. TASM's regret-based approach described in Section IV-C converges to such good layouts over time as queries are executed. Fig. 9 also shows how object densities affect performance.

B. Known queries and known objects

We first present TASM's fundamental video layout optimization assuming a known workload, meaning that TASM knows which objects will be queried, *and* the semantic index contains their locations. These assumptions are unlikely to hold in practice, and we relax them in the next section.

Given a workload and a complete semantic index, TASM decides on SOT boundaries then picks a tile layout for each SOT to minimize execution costs over the entire workload. More formally, the goal is to partition a video into SOTs, $v = s_0 \oplus \cdots \oplus s_n$ and find $\mathscr{L}^* = \arg\min_{\mathscr{L}} C(v, Q, \mathscr{L})$.

The experiment in Fig. 10 motivates us to create small SOTs because they perform best. We therefore partition the video such that each GOP corresponds to a SOT in the tiled video. This produces a tiled video with a similar storage cost as the untiled video because it has the same number of keyframes.

It would be too expensive for TASM to consider every possible layout, uniform and non-uniform, for a given SOT. However, tile layouts that isolate the queried objects should improve performance the most. Additionally, we empirically demonstrate that non-uniform layouts outperform uniform layouts (see Fig. 7a), and that fine-grained layouts outperform coarse-grained layouts (see Fig. 9). Therefore, for each s_i, TASM only considers a fine-grained, non-uniform layout around the objects targeted by queries in that SOT, $O_{s_i} \subseteq O_Q$.

TASM's optimization process proceeds in two steps. First, for each s_i and associated layout, $L{=}\textsc{partition}(s_i, O_{s_i})$, TASM estimates if re-tiling the SOT with L will improve query performance at all. As described in Section III-C4, TASM does not tile s_i when $P(s_i, Q, L){>}\alpha{\cdot}P(s_i, Q, \omega)$, where α specifies how much a tile layout must reduce the amount of decoding work compared to an untiled video (i.e., $L{=}\omega$). In our

1: $O_{Q'} \leftarrow \emptyset$, $L_{alt} \leftarrow \emptyset$, $\forall s_j \in v : \delta^j \leftarrow 0, L_0^j \leftarrow \omega$
2: **for all** $q_i \in Q$ **do**
3: $O_{Q'} \leftarrow O_{Q'} \cup O_{q_i}$
4: $L'_{alt} = \mathcal{P}(O_{Q'})$
5: **for all** $L_k \in L'_{alt} - L_{alt}$ **do**
6: **for** $m = 0, \ldots, i-1$ **do**
7: $\forall s_j \in v : \delta_k^j \leftarrow \delta_k^j + \Delta(q_m, L_m^j, L_k)$
8: $L_{alt} \leftarrow L'_{alt}$
9: **for all** $L_k \in L_{qlt}$ **do**
10: $\forall s_j \in v : \delta_k^j \leftarrow \delta_k^j + \Delta(q_i, L_i^j, L_k)$
11: **for all** $s_j \in v$ **do**
12: $k^* \leftarrow arg\,max_k \delta_k^j$
13: **if** $\delta_{k^*}^j > \eta \cdot R(s_j, L_{k^*})$ **then**
14: Retile s_j with L_{k^*}. $\delta^j \leftarrow 0$

Fig. 6. Pseudocode for incrementally adjusting layouts

experiments we find α=0.8 to be a good threshold. As shown in Fig. 11, this value of α prevents TASM from picking tile layouts that would slow down query processing, but does not cause it to ignore layouts that would have significantly sped up queries. Second, from among all such layouts, TASM selects the layout with the smallest estimated cost for the workload.

C. Unknown queries and unknown objects

In practice, objects targeted by queries and their locations are initially unknown. Physically tuning the tile layout is then similar to the online index selection problem in relational databases [18]. In both, the system reorganizes physical data or builds indices with the goal of accelerating unknown future queries. However, while a nonclustered index can benefit queries over relational data because there are many natural random access points, video data requires physical reorganization to introduce useful random access opportunities. As TASM observes queries and learns the locations of objects, it makes incremental changes to the video's layout specification to introduce these random access points.

TASM optimizes the layout of each SOT independently because each SOT's contribution to query time and the cost to re-encode it are independent of other SOTs. TASM optimizes the layout of an SOT based on the queries that have targeted it so far. TASM may even tile it multiple times with different layouts as the semantic index gains more complete information and TASM observes queries that target additional objects.

As TASM re-encodes portions of the video, the SOT s_j transitions through a series of layouts: $\mathbf{L}$=$[L_0^j, \cdots, L_n^j]$. TASM's goal is to pick a sequence of layouts that minimizes the total execution cost over the workload by finding $\mathbf{L}^*$=$\arg\min_{\mathbf{L}} \sum_{q_i \in Q}(C(s_j, q_i, L_i^j) + R(s_j, L_i^j))$. The first term measures the cost of executing the query with the current layout, and the second term measures the cost of transitioning the SOT to that layout. If the layout does not change (i.e., L_{i-1}^j=L_i^j), then $R(s_j, L_i^j)$=0. However, future queries are unknown, so TASM must pick L_{i+1}^j without knowing q_{i+1}. Therefore, TASM uses heuristics to pick a sequence of layouts, $\hat{\mathbf{L}}$, that approximates $\mathbf{L}^*$. While there are no guarantees on how close $\hat{\mathbf{L}}$ is to $\mathbf{L}^*$, we show in Section V-C that empirically these layouts perform well. One such heuristic is guided by the observation that many applications query for similar objects over time. TASM therefore creates layouts optimized for objects it has seen so far. More formally, let $O_{Q'}$ be the set of objects from Q'=$(q_0, \cdots, q_i) \subseteq Q$. TASM only considers non-uniform layouts around objects in $O_{Q'}$ for L_{i+1}.

Now consider a future query q_j that targets a new class of object: $O_{q_j} \not\subseteq O_{Q'}$. While L_{i+1} will not be optimized for O_{q_j}, TASM attempts to create layouts that will not hurt the performance of queries for new types of objects. It does this by creating fine-grained tile layouts because, as shown in Fig. 9, fine-grained tiles lead to better query performance than coarse-grained tiles when queries target new types of objects (PARTITION$(s, O'), O' \cap O_{q_j} = \emptyset$). Objects that are not considered when designing the tile layout may intersect multiple tiles, and it is more efficient for TASM to decode all intersecting tiles when the tiles are small, as in fine-grained layouts, than when the tiles are large, as in coarse-grained layouts.

At a high level, TASM tracks alternative layouts based on the objects targeted by past queries and identifies potentially good layouts from this set by estimating their performance on observed queries. TASM's incremental tiling algorithm builds on related regret-minimization techniques [18], [19]. Regret captures the potential utility of alternative indices or layouts over the observed query history when future queries are unknown. As each query executes, TASM accumulates regret δ_k^j for each SOT s_j and alternative layout L_k, which measures the total estimated performance improvement compared to the current tile layout over the query history.

Fig. 6 shows the pseudocode of our core algorithm for incremental tile layout optimization using regret minimization. Initially, TASM has not seen queries for any objects, so it does not have any alternative layouts to consider, and each SOT is untiled (line 1). After each query, TASM updates the set of seen objects and alternative layouts (lines 3-4). Each potential layout is a subset of the seen objects that have location information in the semantic index. TASM then accumulates regret for each potential layout by computing Δ and adding it to δ. Δ measures the estimated performance improvement of executing the query with an alternative layout rather than the current layout, using the cost function described in Section IV-A: $\Delta(q, L, L') = C(s, q, L) - C(s, q, L')$. Layouts with high Δ values would likely reduce query costs, while layouts with low or negative values could hurt query performance. TASM accumulates these per-query Δ's into regret to estimate which layouts would benefit the entire query workload.

TASM first retroactively accumulates regret for new layouts based on the previous queries (lines 5-7), and then accumulates regret for the current query (lines 9-10). Finally, TASM weighs the performance improvements against the estimated cost of transitioning a SOT to a new layout. In lines 11-14, TASM only re-tiles s_j once its regret exceeds some proportion of its estimated retiling cost: $\delta_k^j > \eta \cdot R(s_j, L_k)$.

As an example, consider a city planning application looking through traffic videos for instances where both cars and pedestrians were in the crosswalk at the same time. Initially the traffic video is untiled, so for each s_i, $\mathscr{L}(s_i)$=ω. Suppose the

first query requests cars in s_0. TASM updates L_{alt}=$\{\{car\}\}$ to consider layouts around cars. TASM accumulates regret for s_0 as δ^0_{car}=$\Delta(q_0, \omega, \text{PARTITION}(s_0, \{car\}))$, and it is positive because tiling around cars would accelerate the query. Suppose the next query is for people in s_0. TASM updates L_{alt}=$\{\{car\}, \{person\}, \{car, person\}\}$ to consider layouts around cars and people. The regret for $\text{PARTITION}(s_0, \{car\})$ on q_1 will likely be negative because layouts around anything other than the query object tend to perform poorly (see Fig. 9b), so δ^0_{car} decreases. TASM retroactively accumulates regret for the new layouts. The accumulated regret for $\text{PARTITION}(s_0, \{person\})$ will be similar to δ^0_{car} because it would accelerate q_1 and hurt q_0. $\text{PARTITION}(s_0, \{car, person\})$ has positive regret for both q_0 and q_1, so after both queries it has the largest accumulated regret.

The threshold η (see line 13) determines how quickly TASM re-tiles the video after observing queries for different objects. Using $\eta = 0$ risks wasting resources to re-tile SOTs. The work to re-tile could be wasted if a SOT is never queried again because no queries will experience improved performance from the tiled layout. The work to re-tile can also be wasted if queries target different objects because TASM will re-tile after each query with layouts optimized for just that query. Values of $\eta > 0$ enable TASM to observe multiple queries before picking layouts, so the layouts can be optimized for multiple types of objects. Observing multiple queries before committing to re-tiling also enables TASM to avoid creating layouts optimized for objects that are infrequently queried because layouts around more representative objects will accumulate more regret. However, if the value of η is too large, it reduces the number of queries whose performance benefits from the tiled layout. Using a value of $\eta = 1$ is similar to the logic used in the online indexing algorithm in [18], and we find it generally works well in this scenario, as shown in Fig. 12. If the types of objects queries target changes, this incremental algorithm will take some amount of time to adjust to the new query distribution, depending on the value of η.

D. ROI tiling

Initially, nothing is known about a video. As we discussed in Section I, in many systems, the first object detection query performs a full scan and applies a simple predicate to filter away uninteresting frames or compute statistics. Because of the speed of these initial filters, decoding and preprocessing is the bottleneck for this phase [13]. To accelerate this full scan phase, TASM also uses predicate pushdown. Instead of creating tiles around objects, however, TASM creates tiles around more general *regions of interest* (ROIs), where objects are expected to be located. ROIs are defined by bounding boxes, so TASM uses the same tiling strategies described in previous sections. TASM accepts a user-defined predicate that detects ROIs and inserts the associated bounding boxes into TASM's semantic index. Examples include applying background subtraction to identify foreground objects, running specialized models trained to identify a specific object type [7], [12], extracting motion vectors to isolate areas with moving objects, or any other inexpensive computation. More expensive predicates may also be used by applying them every n frames, as in [11].

Generating ROIs and creating tiles around these regions are operations that a compute-enabled camera can perform directly as it first encodes the video. Cameras are now capable of running these lightweight predicates as video is captured [26]. For example, specialized background subtractor modules can run at over 20 FPS on low-end hardware [27]. This optimization is designed to be implemented on the edge.

Through its semantic predicate pushdown optimization, TASM improves the performance of object detection queries by only decoding tiles that contain ROIs. As we show in Section V-E, an initial ROI layout in combination with semantic predicate pushdown can significantly accelerate the full scan phase of object detection queries while maintaining accuracy.

TABLE I
VIDEO DATASETS

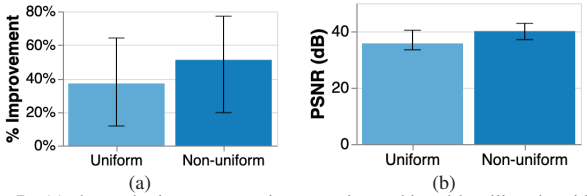
Video dataset	Duration (sec.)	Res.	Per-frame object coverage (%)	Frequently occurring objects
Visual Road [28]†	540–900	2K, 4K	0.06–10	car, person
Netflix public [29]	6	2K	0.32–49	person, car, bird
Netflix OS [30]*	720	2K, 4K	25–45	person, car, sheep
XIPH [31]	4–20	2K, 4K	2–59	car, person, boat
MOT16 [32]	15–30	2K	3–36	car, person
El Fuente [33]	480 (full)	4K	1–47	person, car,
	15–45 (scenes)			boat, bicycle

† Synthetic videos * Both real and synthetic videos

V. EVALUATION

We implemented a prototype of TASM in C++ integrated with LightDB [24]. TASM encodes and decodes videos using NVENCODE/NVDECODE [34] with the HEVC codec. We perform experiments on a single node running Ubuntu 16.04 with an Intel i7-6800K processor and an Nvidia P5000 GPU. Our prototype does not parallelize encoding or decoding multiple tiles at once. We use FFmpeg [35] to measure video quality.

We evaluate TASM on both real and synthetic videos with a variety of resolutions and contents as shown in Table I. Visual Road videos simulate traffic cameras. They include stationary videos as well as videos taken from a roof-mounted camera (the latter created using a modified Visual Road generator [28]) The Netflix datasets primarily show scenes of people. The XIPH dataset captures scenes ranging from a football game to a kayaker. The MOT16 dataset contains busy city scenes with many people and cars. The El Fuente video contains a variety of scenes (city squares, crowds dancing, car traffic). In addition to evaluating the full El Fuente video, we also manually decompose it into individual scenes using the scene boundaries specified in [33] and evaluate each independently. We do not evaluate on videos with resolution below 2K because we found that decoding low-resolution video did not exhibit significant overhead. All experiments populate the semantic index with object detections from YOLOv3 [36], except for the MOT16 videos where we use the detections from the dataset [32]. We store the semantic index using SQLite [37], and TASM maps bounding boxes to tiles at query time.



(a) (b)

Fig. 7. (a) shows the improvement in query time achieved by tiling the video using the fastest uniform and non-uniform layout for each video and query object. (b) shows the quality of these layouts compared to the untiled video.

The queries used in the microbenchmarks evaluated in Section V-A and V-B are subframe selection queries of the form "`SELECT o FROM v`", which cause TASM to decode all pixels belonging to object class `o` in video `v`. The queries used in the workloads in Section V-C additionally include a temporal predicate (i.e., "`SELECT o FROM v WHERE start < t < end`"). [2] Reported query times include both the index look-up time and the time to read from disk and decode the tiles.

Unless otherwise specified, queries target the most frequently occurring objects in each video. When videos primarily show a single type of object (e.g., some Netflix public dataset videos show only people), queries target just that object. When videos feature multiple types of objects with similar frequency (e.g., the Visual Road videos show similar numbers of cars and people), we evaluate on queries that target each object type. Queries over the MOT16 videos retrieve cars *and* people because the bounding boxes that come with the dataset are unlabeled, so we store them in the semantic index with a generic label of "object". For all graphs, the bars show the median value across videos, while the error bars denote the interquartile range (IQR) across videos. The performance differs across videos because they have different object densities, which affects TASM's efficacy as described in Section V-B2. However, the runtime for a single query on any video has low variance. The standard deviation for multiple executions of the same query is $< 1\%$ of that query's mean execution time.

A. Tiling effect on decode cost and quality

We first evaluate whether tiling can provide meaningful improvements in query time without degrading the visual quality of videos. We find that non-uniform layouts yield better query performance and higher video quality than uniform layouts. Fig. 7 only shows results for videos and queries that benefit from tiling, using the layouts that empirically led to the greatest performance improvement. We discuss how TASM determines whether to tile a video in Section V-B3 and how it selects the optimal tile layout in Section V-B and Section V-C.

Fig. 7a shows the improvement in query time achieved by operating over a tiled video compared to a video that is not tiled. For a given video and query object, a non-uniform layout provides an average of 10% improvement and up to a 35% improvement over the best uniform layout.

[2]While we use SQL to explain the experiments because of its familiarity to most readers, other language bindings on TASM's API are possible; the language itself is not the focus of this paper.

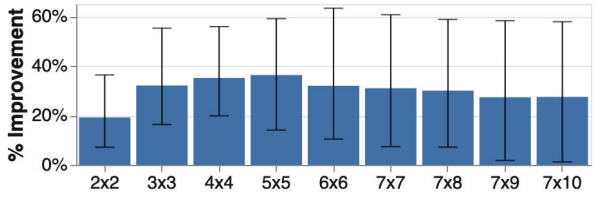


Fig. 8. This figure shows improvement in query time achieved with various uniform layouts compared to the untiled video.

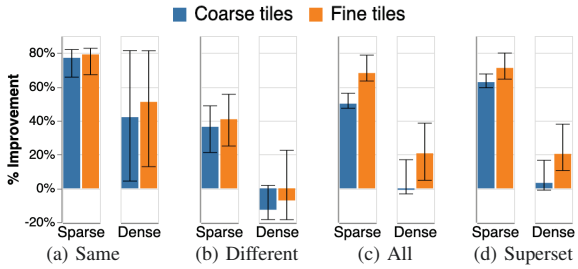


Fig. 9. The effect of tile granularity on query time compared to untiled videos. All videos used a one second tile layout duration. Objects occupy $<$20% of each frame on average in "sparse", and $\geq$20% in "dense" videos.

Fig. 7b shows that tiling maintains good visual quality when the tiles are stitched to recover the full frame. We measure quality using peak signal-to-noise ratio (PSNR), where values above 30 dB are acceptable [38], and videos with values $\geq$ 40 dB are perceived to have good quality [29], [39]. PSNR was computed over the entire tiled video stitched using homomorphic stitching [24] and compared against the untiled video. For comparison, the median PSNR after re-encoding the videos without tiles is 46 dB. Non-uniform layouts achieve an average PSNR of 40 DB, while uniform layouts have an average of 36 dB. PSNR is likely lower for the uniform layouts because the layouts with the largest performance improvement have many tiles (the median number of tiles is 25), and therefore a large number of tile boundaries where quality is degraded.

B. Microbenchmarks

1) Uniform tiles: We dig deeper into the results of Fig. 7 and show in Fig. 8 the performance improvements when varying the number of uniform tiles on the same set of videos. We increase the number of uniform tiles first by increasing the number of rows and columns together, and then by only increasing the number of columns once the height of each tile reached the minimum height allowed by the decoder. Fig. 8 shows that creating more uniform tiles initially improves query time because tiles contain fewer non-object pixels. However, as the number of tiles grows, the per-tile decode overhead begins to slow queries down. Additionally, variation in performance across videos and queries increases with the number of tiles, as indicated by the widening IQR bars, demonstrating that the same uniform layout does not work equally well on all videos and queries.

2) Non-uniform tiles: The performance of non-uniform layouts depends on the objects queries target and the objects considered when designing the tile layout. Fig. 9 shows results from different settings. We classify layouts as *same*, *different*, *all*, or *superset*. "Same" describes a tile layout around the query object. "Different" describes a layout around an object different

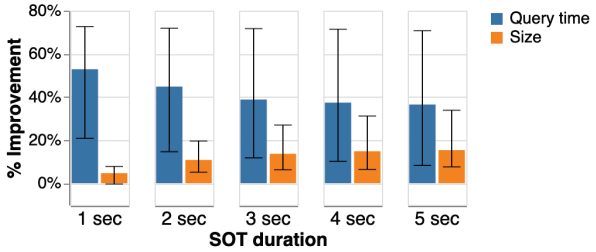


Fig. 10. This plot shows the effect of SOT duration on query time and storage cost. Tiled videos were encoded with fine-grained tiles and a GOP length equal to the SOT duration.

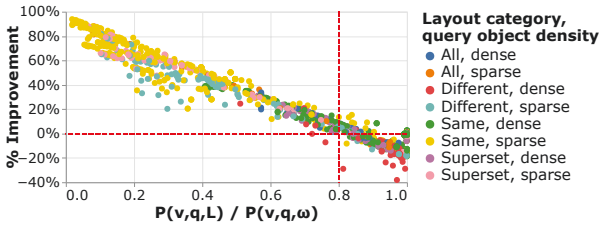


Fig. 11. Ratio of the number of pixels decoded with a non-uniform layout to the number decoded without tiles vs. performance improvement. Each point represents a video, query object, and non-uniform layout. Points below the horizontal line at 0% represent cases where queries ran more slowly on the tiled video. Points to the right of the vertical line at 0.8 represent videos that would not be tiled when the threshold for tiling requires the ratio to be < 0.8.

from the query object (e.g., tiling around people but querying for cars). "All" describes tiling around *all* objects detected in the video. Finally, "superset" evaluates tiling around the target object and only 1-2 other, frequently occurring objects (e.g., tiling around cars *and* people, as in Fig. 3b). We further classify videos as *sparse*, where the average area occupied by all objects in a frame is $<20\%$, or *dense*, where it is $\geq 20\%$. Fig. 9 shows the results. The "different" and "superset" categories only use Visual Road videos and El Fuente scenes that feature multiple object classes; the other videos have a single primary object type.

Fig. 9 shows that tiling generally improves performance in sparse videos more than dense videos, and tile granularity has the largest impact when objects are dense. Fig. 9a shows that when the tile layout is constructed around the query object, both coarse- and fine-grained tiles significantly improve query performance. Fig. 9b shows that tiling around an object type different from the query object hurts performance when objects are dense. This happens when one object is more dense than the others. Querying for the dense object using a layout around the sparse object requires TASM to decode most of the tiles because the dense object occupies much of each frame. Querying for a sparse object using a layout around the dense object also requires most of the frame to be decoded because tiles around dense objects tend to be large. TASM avoids creating these ineffective layouts around dense objects using the decision rule from Section IV-B, which we evaluate in Section V-B3. Improvement in sparse videos is reduced, but still positive; although the query object may intersect multiple tiles, TASM still performs less work if the tiles are small.

Fig. 9c shows that tiling around all objects is effective only when objects are sparse. When objects are dense, median improvement is 1% *worse* for coarse-grained tiles. Fig. 9d shows that the "superset" strategy performs similarly to "all"; considering only two or three types of objects rather than all objects when designing layouts achieves small performance gains.

These results show that tiling around anything other than the query object slows queries down compared to tiling around the query object. However, fine-grained tiles can still lead to moderate performance improvements in these cases because they are smaller, so fewer non-object pixels must be decoded.

Sequence of tiles (SOT) duration. Here we evaluate the impact of SOT duration (the number of frames with the same layout) on the performance of non-uniform tile layouts. SOT duration affects the sizes of both tiles and the video. Layout changes must happen at GOP boundaries, so short SOTs require short GOPs and lead to larger storage sizes (see Section II).

Fig. 10 shows the effect of SOT duration on query performance and storage size. The tiled videos are encoded with a GOP length equal to the SOT duration. We compare query performance and storage size to an untiled video encoded with one-second GOPs (the default GOP duration in most video encoders). Shorter SOT durations lead to larger improvements in query performance because the tiles are smaller and contain fewer non-object pixels. However, shorter SOTs lead to larger storage costs because there are more keyframes. Note that we see a small improvement in the size of the tiled video with one-second SOTs compared to the original video (also encoded with one-second GOPs); this is due to video encoders being inherently lossy and having the ability to exploit additional compression opportunities during recompression. These results demonstrate that setting SOT duration to GOP length is optimal since it leads to the best performance without storage overhead.

3) Not tiling: There are videos where tiling is an ineffective strategy to improve query performance. To identify cases where tiling should not be used, we evaluate the effectiveness of a decision rule based on the number of pixels decoded with a given layout. Fig. 11 plots the improvement in query time against the ratio of pixels decoded with a non-uniform layout compared to the untiled video (i.e., $P(v,q,L)/P(v,q,\omega)$) for various videos and query objects. The figure includes a sampling of diverse layouts, both optimal and suboptimal. The "same" category includes the greatest variety of layouts measured, including suboptimal layouts. While many points overlap, the key observation is that queries for sparse objects primarily lie in the top-left quadrant. This aligns with the expected improvements based on the cost function described in Section IV-A. Using a threshold of not tiling when $P(v,q,L)/P(v,q,\omega)>0.8$ captures nearly all tile layouts that slow queries down (i.e., the improvement is negative). A small number of videos achieve minor performance improvements ($<20\%$) above this threshold (the upper-right quadrant).

C. Incremental tiling

We next evaluate strategies for incremental tiling over various subframe selection workloads, which we construct to represent possible query patterns over videos. The baseline strategies are not tiling the video ("Not tiled") and tiling around all detected objects before queries are processed ("All objects").

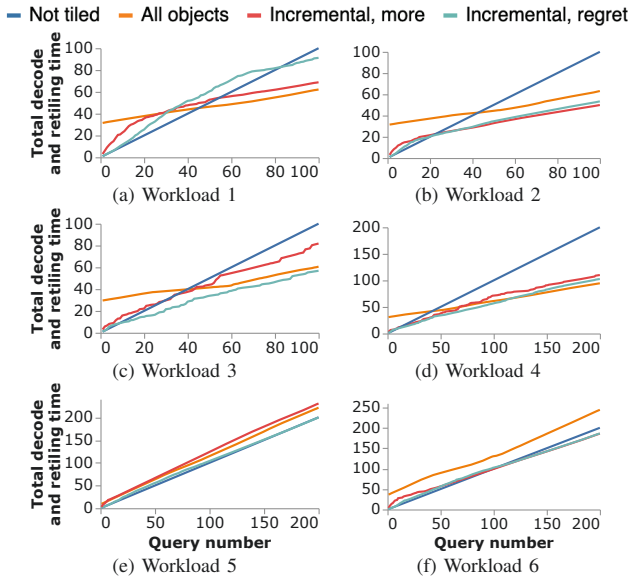


Fig. 12. Cumulative decode and re-tiling time for various workloads. Values are normalized to the time to execute each query over the untiled videos.

We compare against two incremental strategies. "Incremental, more" re-tiles each GOP with a non-uniform, fine-grained layout around all object classes that have been queried so far. For example, if a GOP were queried for cars and then people, TASM would first tile around cars and then re-tile around cars and people. Finally, we evaluate the regret-based approach from Section IV-C ("Incremental, regret"). In this strategy, TASM tracks alternative layouts based on the objects queried so far, and re-tiles GOPs once the regret for a layout exceeds the estimated re-encoding cost if the layout is not expected to hurt performance.

TASM estimates the layout will hurt performance if, for any query, $P(s_i, q_i, L) \geq \alpha \cdot P(s_i, q_i, \omega)$, where α=0.8 (see Section IV-B). TASM estimates the regret using the cost function described in Section IV-A. Similarly, the re-encoding cost is estimated using a linear model based on the number of pixels being encoded. It was fit based on the time to encode videos with the various layouts used in the microbenchmarks.

As we are focused on the operations at the storage level, we measure the cumulative time to read video from disk and decode it to answer each query, and re-tile it with new layouts as needed. The time to initially tile the video around all objects is included with the first query for the "all objects" strategy. We normalize each query's cost to the time to execute that query on the untiled video, so each query with the "not tiled" strategy has a cost of 1. The lines in Fig. 12 show the median over all videos the workload was evaluated on. We evaluate the first four workloads on Visual Road videos, which have sparse objects, and the last two on videos and scenes with dense objects.

As Fig. 12 shows, the regret-based approach consistently performs best across all evaluated methods, except for Workload 1. TASM's regret-based approach was designed for more dynamic workloads than Workload 1 where the same query is evaluated across the entire video. For this type of workload, running object detection and tiling up front is a reasonable strategy because all of the results will be used.

We now drill down in the results of each workload. Queries in Workload 1 target a single object class across the entire video. The workload consists of 100 one-minute queries for cars uniformly distributed over each Visual Road video. As shown in Fig. 12a and discussed above, pre-tiling around all objects performs well when queries target the entire video. Incrementally tiling without regret also performs well because all queries target the same object, so SOTs are re-tiled to a layout that speeds up future queries. The regret-based approach performs poorly over a small number of queries because TASM must observe multiple queries over the same SOT before enough regret accumulates to re-tile. This requires many total queries to be executed when they are uniformly distributed over the entire video.

We next evaluate Workload 2, which examines the performance when queries are restricted to a subset of the video. Workload 2 consists of 100 one-minute queries over the first 25% of each Visual Road video. Each query has a 50% chance of being for cars or people. As shown in Fig. 12b, both incremental strategies perform similarly well. Both outperform pre-tiling the entire video around all objects, which is wasteful when only a small portion of the video is ever queried.

Workload 3 measures the performance when queries are biased towards one section of a video and particular object types. It consists of 100 queries over the Visual Road videos, where each query has a 47.5% chance of being for cars or people, and a 5% chance of being for traffic lights. We exclude one 4K video that did not contain a traffic light. The start frame of each query is picked following a Zipfian distribution, so queries are more likely to target the beginning of the video. As shown in Fig. 12c, the regret-based approach performs better than incrementally tiling around more objects because it spends less time re-tiling sections of the video with tile layouts designed around the rarely-queried object.

Workload 4 measures performance when queries target different objects over time. It consists of 200 one-minute queries following a Zipfian distribution over the Visual Road videos. The middle third of the queries target people, and the rest target cars. As shown in Fig. 12d, the incremental, regret-based approach performs well and does not exhibit large jumps in decode and re-tiling time when the query object changes.

Workload 5 measures performance when tiling is not effective. It is evaluated on select videos from the Xiph, Netflix public dataset, and scenes from the El Fuente video that contain diverse scenes with many types of objects (e.g., markets with people, cars, and food). The queries are uniformly distributed, and each randomly targets one of the video's primary objects within one-second. As shown in Fig. 12e, only the regret-based approach keeps costs similar to not tiling. "All objects" performs poorly because objects are dense in these scenes. "Incremental, more" performs poorly because it spends time re-tiling with layouts that perform similarly to the untiled video.

Finally, Workload 6 measures performance when tiling around the query object is beneficial, but tiling around all objects is not. It is evaluated on select videos from the Netflix public dataset and scenes from the full El Fuente video that fit

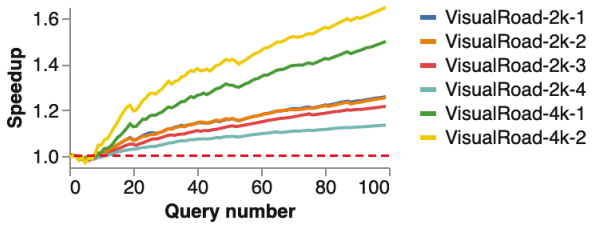


Fig. 13. Speedup achieved with TASM over the Visual Road object detection workload. The lines show the median speedup over six orderings of the queries.

this criteria. The queries are uniformly distributed, and each targets the same object class over one second. As shown in Fig. 12f, both incremental strategies eventually achieve layouts that perform better than not tiling. "All objects" performs poorly because objects in these videos are dense.

D. Macrobenchmark

Beyond the decoding benchmarks, we also evaluate TASM's performance on an end-to-end workload from the Visual Road benchmark [28], specifically Q7. Each query in the workload specifies a temporal range and a set of object classes. The following tasks are executed per-query: (i) detect objects if not previously done on the specified temporal range, (ii) draw boxes around the specified object classes, and (iii) encode the modified frames. The original Visual Road query involves masking the background pixels, but we omit that step to demonstrate TASM's benefits when users want to view full frames. We compare the performance of executing this query on untiled frames to TASM with incremental, regret-based tiling. We detect objects by running YOLOv3 [36] every three frames. TASM adds bounding boxes by decoding only the tiles that contain the requested objects, drawing the boxes, then re-encoding these tiles. TASM outputs the full frame by homomorphically stitching the modified tiles that contain the object with the original tiles that do not contain the object.

We execute 100 one-minute queries over the Visual Road videos, using a Zipfian distribution over time-ranges. Each query is randomly for cars or people. Fig. 13 shows the median speedup achieved with TASM compared to the untiled video over six orderings of the queries. TASM reduces the total workload runtime by 12-39% across the videos. Object detection contributes significantly to the total runtime and LightDB does not use a pre-filtering step to accelerate this operation. If we examine one instance of the workload where the last 20 queries no longer need to perform object detection and execute after TASM has found good layouts, the median improvement for these queries ranges from 23% to 66% across the videos. While these queries request the full frame, TASM accelerates them by processing just the relevant regions of the frame, which allows it to decode and encode less data.

E. Object detection acceleration

We now evaluate TASM's ability to accelerate the full scan phase of object detection queries, as described in Section I. One system that uses specialized models during the full scan phase is BlazeIt [3]. For example, it uses a specialized counting model to compute aggregates. We evaluate TASM's ability to accelerate this phase using BlazeIt's counting model as a representative fast model. This model runs at over 1K frames per second (fps), while preprocessing the frames runs below 300 fps. TASM reduces the preprocessing bottleneck and achieves up to a 2× speedup while maintaining the model's accuracy.

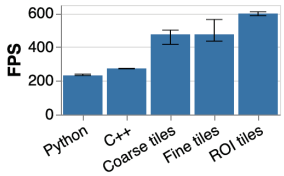


Fig. 14. Specialized model preprocessing throughput

TABLE II
MODEL ACCURACY

	Day 1	Day 2	Day 3
Full	0.79	0.51	0.56
ROI	0.84	0.61	0.51
Coarse	0.76	0.60	0.54

The preprocessing phase includes reading video from disk, decoding and resizing frames, normalizing pixel values, and transforming the pixel format. BlazeIt implements this using Python, OpenCV [40], and Numpy [41] ("Python" in Fig. 14). We reimplemented this using C++, NVDECODE [34], and Intel IPP [42] to fairly compare against TASM ("C++"). We evaluate on three days of BlazeIt's `grand-canal` video dataset.

We compare against using semantic predicate pushdown with ROI layouts generated by TASM. We first use MOG2-based background segmentation implemented in OpenCV [40] to detect foreground ROIs on the first frame of each GOP. This is a throughput that recent mobile devices are known to operate above [27], and therefore it would be possible for this step to be offloaded to a compute-enabled camera as discussed in Section IV-D. We use TASM to create fine-grained tiles ("Fine tiles") and coarse-grained tiles ("Coarse tiles") around the foreground regions. We also compare against a tile layout created around a manually-specified ROI capturing the canal in the lower-left portion of each frame ("ROI").

Fig. 14 shows the preprocessing throughput when operating on entire frames compared to just the tiles that contain ROIs. Operating on tiles improves throughput by up to 2× and therefore reduces the bottleneck for performing inference with the specialized model. We next verify that using tiles rather than full frames does not negatively impact the model's accuracy. We use the same model architecture for tiled inputs. However, rather than training and inferring using full frames, we use a single tile from each frame that contains all ROIs. For each strategy we train BlazeIt's counting model on the first 150K frames or tiles from the first day of video. We evaluate this model on 150K frames or tiles from each day (using a different set of frames for the first day). As shown in Table II, models trained and evaluated on tiles show similar accuracy to full frame training within each day.

VI. Related Work

As mentioned in Section I, many systems optimize extracting semantic content from videos. BlazeIt [3] and NoScope [7] apply specialized NNs that run faster than general models. Other systems filter frames before applying expensive models: probabilistic predicates [12] and ExSample [43] use statistical techniques, MIRIS [11] uses sampling, and SVQ [44] and *IC* and *OD* filters [45] use deep learning filters. These systems and

techniques can use TASM to run models on specific ROIs to reduce their preprocessing overhead. Focus [9] shifts some processing to ingest-time. Systems such as LightDB [24], Optasia [1], and Scanner [8] accelerate queries through parallelization and deduplication of work, while VideoEdge [46] distributes processing over clusters. These general VDBMSs could incorporate TASM to further accelerate performance. Panorama [6] and Rekall [47] expand the set of queries that can be executed over videos, which is orthogonal to video storage.

Other systems also target storage-level optimizations. VStore [10] modifies encoding parameters to accelerate processing while maintaining accuracy. Smol [13] jointly optimizes video resolution and NN architectures to achieve high accuracy while accelerating preprocessing, but, like VStore, only considers reducing the resolution of videos while TASM maintains video quality. Vignette [48] uses tiles for perception-based compression but only considers uniform layouts.

TASM's incremental tiling approach is inspired by database cracking [16], [17], which incrementally reorganizes the data processed by each query, and online indexing [18] which creates and modifies indices as queries are processed. Regret has also been used to design an economic model for self-tuning indices in a shared cloud database [19]. TASM extends these relational storage techniques to provide efficient access to video data.

Other application domains have observed the usefulness of retrieving spatial subsets of videos. The MPEG DASH SRD standard [49] is motivated by a similar observation that video streaming clients occasionally request a spatial subset of videos. While it specifies a model to support streaming spatial subsets, it does not specify *how* to efficiently partition videos into tiles.

VII. Conclusion

We presented TASM, a tile-based storage manager that accelerates subframe selection and object detection queries by targeting spatial frame subsets. TASM incrementally tiles sections of the video as queries execute, leading to improved performance (up to 94%). We also showed how TASM alleviates bottlenecks by only reading areas likely to contain objects.

Acknowledgments. This work was supported in part by NSF award CCF-1703051, an award from the University of Washington Reality Lab, a gift from Intel, and DARPA through RC Center grant GI18518. A. Mazumdar was also supported in part by a CoMotion Commercialization Fellows grant.

References

[1] Y. Lu *et al.*, "Optasia: A relational platform for efficient large-scale video analytics," in *SoCC*, 2016, pp. 57–70.
[2] H. Zhang *et al.*, "Live video analytics at scale with approximation and delay-tolerance," in *NSDI*, 2017, pp. 377–392.
[3] D. Kang *et al.*, "BlazeIt: Optimizing declarative aggregation and limit queries for neural network-based video analytics," *PVLDB*, vol. 13, 2019.
[4] X. Wang *et al.*, "Skyeyes: adaptive video streaming from uavs," in *HotWireless@MobiCom*, 2016, pp. 2–6.
[5] J. Wang *et al.*, "Bandwidth-efficient live video analytics for drones via edge computing," in *SEC*, 2018, pp. 159–173.
[6] Y. Zhang *et al.*, "Panorama: A data system for unbounded vocabulary querying over video," *PVLDB*, vol. 13, pp. 477–491, 2019.
[7] D. Kang *et al.*, "Noscope: Optimizing deep CNN-based queries over video streams at scale," *PVLDB*, vol. 10, pp. 1586–1597, 2017.
[8] A. Poms *et al.*, "Scanner: efficient video analysis at scale," *ACM Trans. Graph.*, vol. 37, pp. 138:1–138:13, 2018.
[9] K. Hsieh *et al.*, "Focus: Querying large video datasets with low latency and low cost," in *OSDI*, 2018, pp. 269–286.
[10] T. Xu *et al.*, "VStore: A data store for analytics on large videos," in *EuroSys*. ACM, 2019, pp. 16:1–16:17.
[11] F. Bastani *et al.*, "MIRIS: fast object track queries in video," in *SIGMOD*, 2020, pp. 1907–1921.
[12] Y. Lu *et al.*, "Accelerating machine learning inference with probabilistic predicates," in *SIGMOD*. ACM, 2018, pp. 1493–1508.
[13] D. Kang *et al.*, "Jointly optimizing preprocessing and inference for DNN-based visual analytics," *CoRR*, vol. abs/2007.13005, 2020.
[14] Waymo, "Waymo open dataset FAQ," waymo.com/open/faq, 2021.
[15] M. AbuTaha *et al.*, "End-to-end real-time ROI-based encryption in HEVC videos," in *EUSIPCO*. IEEE, 2018, pp. 171–175.
[16] S. Idreos *et al.*, "Database cracking," in *CIDR*, 2007, pp. 68–78.
[17] F. Halim *et al.*, "Stochastic database cracking: Towards robust adaptive indexing in main-memory column-stores," *PVLDB*, vol. 5, 2012.
[18] N. Bruno *et al.*, "An online approach to physical design tuning," in *ICDE*, 2007, pp. 826–835.
[19] D. Dash *et al.*, "An economic model for self-tuned cloud caching," in *ICDE*, 2009, pp. 1687–1693.
[20] "Advanced video coding for generic audiovisual services," Rec. ITU-T H.264 and ISO/IEC 14496-10, 06 2019.
[21] "High efficiency video coding," Rec. ISO/IEC 23008-2, Nov 2019.
[22] P. de Rivaz *et al.*, "AV1 bitstream & decoding process specification," *The Alliance for Open Media*, p. 182, 2018.
[23] G. J. Sullivan *et al.*, "Overview of the high efficiency video coding (HEVC) standard," *TCSVT*, vol. 22, pp. 1649–1668, 2012.
[24] B. Haynes *et al.*, "LightDB: A DBMS for virtual reality video," *PVLDB*, vol. 11, pp. 1192–1205, 2018.
[25] S. Chaudhuri *et al.*, "Autoadmin 'what-if' index analysis utility," in *SIGMOD*, 1998, pp. 367–378.
[26] "Bosch camera trainer with fw 7.10 and cm 6.20," boschsecurity.com/us/en/solutions/video-systems/video-analytics/technical-documentation-for-video-analytics, 2019, accessed: 2020-06.
[27] S. Zeevi, "BackgroundSubtractorCNT," https://sagi-z.github.io/BackgroundSubtractorCNT, 2016, accessed: 2021-01-21.
[28] B. Haynes *et al.*, "Visual Road: A video data management benchmark," in *SIGMOD*, 2019, pp. 972–987.
[29] Z. Li *et al.*, "Toward a practical perceptual video quality metric," https://netflixtechblog.com/653f208b9652, 06 2016, accessed: 2020-06-01.
[30] A. Schuler *et al.*, "Engineers making movies (AKA open source test content)," netflixtechblog.com/f21363ea3781, 2018, accessed: 2020-06.
[31] "Xiph.org video test media," media.xiph.org/video/derf, 2019.
[32] A. Milan *et al.*, "MOT16: A benchmark for multi-object tracking," *CoRR*, vol. abs/1603.00831, 2016.
[33] I. Katsavounidis, "Netflix – "El Fuente" video sequence details and scenes," cdvl.org/documents/ElFuente_summary.pdf, accessed: 2020-06.
[34] "Nvidia video codec," developer.nvidia.com/nvidia-video-codec-sdk.
[35] F. Bellard, "FFmpeg," ffmpeg.org, 2018.
[36] J. Redmon *et al.*, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018.
[37] "Sqlite," https://sqlite.org/index.html, 2020.
[38] A. Lottarini *et al.*, "vbench: Benchmarking video transcoding in the cloud," in *ASPLOS*, 2018, pp. 797–809.
[39] M. Vranjes *et al.*, "Locally averaged psnr as a simple objective video quality metric," in *ELMAR*, vol. 1. IEEE, 2008, pp. 17–20.
[40] OpenCV, "Open Source Computer Vision Library," opencv.org, 2018.
[41] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, 2020.
[42] Intel, "Intel integrated performance primitives," software.intel.com/content/www/us/en/develop/documentation/ipp-dev-reference.
[43] O. Moll *et al.*, "Exsample: Efficient searches on video repositories through adaptive sampling," *CoRR*, vol. abs/2005.09141, 2020.
[44] I. Xarchakos *et al.*, "SVQ: streaming video queries," in *SIGMOD*, 2019.
[45] N. Koudas *et al.*, "Video monitoring queries," in *ICDE*, 2020.
[46] C. Hung *et al.*, "VideoEdge: Processing camera streams using hierarchical clusters," in *SEC*, 2018, pp. 115–131.
[47] D. Y. Fu *et al.*, "Rekall: Specifying video events using compositions of spatiotemporal labels," *CoRR*, vol. abs/1910.02993, 2019.
[48] A. Mazumdar *et al.*, "Perceptual compression for video storage and processing systems," in *SoCC*, 2019, pp. 179–192.
[49] O. A. Niamut *et al.*, "MPEG DASH SRD: spatial relationship description," in *MMSys*, 2016, pp. 5:1–5:8.