Data Storage and Exploration in a Video Data Management System

Maureen Daum

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2023

Reading Committee:

Magdalena Balazinska, Chair

Dan Suciu

Ranjay Krishna

Program Authorized to Offer Degree:
Paul G. Allen School of Computer Science & Engineering

University of Washington

Abstract

Data Storage and Exploration in a Video Data Management System

Maureen Daum

Chair of the Supervisory Committee:
Professor Magdalena Balazinska
Paul G. Allen School of Computer Science & Engineering, University of Washington

Increasingly many scientific and engineering domains rely on video data, which is information dense and relatively easy to collect. At the same time, recent advances in computer vision and machine learning have opened the door to automating analysis of this data. As a result, there has been a resurgence of research and innovation in powerful libraries and data management systems to support users in storing, processing, and querying videos. Current video database management systems (VDBMSs) are optimized to efficiently apply machine learning (ML) models over videos to extract their semantic content and generate query results. However, relatively little attention has been directed towards optimizing VDBMS storage managers, despite the fact that videos are stored in compressed formats due to their large size, and accessing the raw pixel information adds non-negligible latency to queries due to the required decoding step. Further, current VDBMSs are limited by assuming that there exist pretrained models relevant to users' videos and queries; this is not the case for many domain-specific datasets, like those collected for wildlife monitoring. Users that lack a relevant pretrained model are unable to benefit from the data management and query processing capabilities that VDBMSs provide.

This dissertation studies the two challenges identified above. First, we introduce TASM, a tile-based storage manager for VDBMSs. TASM accelerates workloads that operate over spatial subsets of frames by storing videos in such a way that makes it possible to decode only

particular regions of frames. In contrast, current VDBMS storage managers must decode entire frames, even if only particular regions are requested, which is inefficient. TASM uses the video codec feature of tiles to divide frames into independently-decodable regions, and it automatically adjusts the layout of these tiles based on the observed query workload.

Second, we introduce VOCALExplore, which is a system that supports users in exploring large video datasets and efficiently training domain-specific models that can be used to automatically extract the semantic content from unlabeled videos. VOCALExplore provides a high-level API that does not require ML expertise; it automatically navigates decisions around feature extraction and sample selection that are essential for producing a high-quality model. Importantly, VOCALExplore does not require an extensive preprocessing phase that causes the user to wait a significant amount of time before interacting with the system. Instead, VOCALExplore operates in a pay-as-you-go manner and performs processing in the background as the user interacts with it.

While there still exist many open problems surrounding VDBMSs, this thesis takes steps towards making them more efficient and usable by a larger audience.

TABLE OF CONTENTS

Page

List of Figures iii

List of Tables v

Chapter 1: Introduction 1
1.1 Opportunities to optimize video data storage 5
1.2 Early video data exploration 9
1.3 Summary of thesis contributions 12
1.4 Thesis Organization 13

Chapter 2: Background information 14
2.1 Video encoding and storage 14
2.2 Object and activity recognition in videos 17
2.3 Summary 20

Chapter 3: Related Work 21
3.1 Early work on VDBMSs 21
3.2 Recent work on VDBMSs 22
3.3 Related work on video data storage 29
3.4 Related work on video data exploration 31

Chapter 4: TASM: A Tile-Based Storage Manager for Video Analytics 35
4.1 Tile-based storage manager design 39
4.2 Tiling strategies 45
4.3 Evaluation 53
4.4 Summary 68

Chapter 5: VOCALExplore: Pay-as-You-Go Video Data Exploration and Model Building 69
5.1 System Overview 73
5.2 Active Learning Manager 77
5.3 Task Scheduler 87
5.4 Evaluation 92
5.5 Summary 104

Chapter 6: VOCALExplore: interface and extensions 108
6.1 VOCALExplore interface 108
6.2 Extensions to VOCALExplore 112
6.3 Extending VOCALExplore to utilize multimodal features 115

Chapter 7: Conclusions & Future Directions 128

Bibliography 132

Appendix A: Frequency-based hypothesis test 147

LIST OF FIGURES

Figure Number		Page
1.1	Architecture of a generic database management system (DBMS).	3
1.2	Video processing pipeline.	6
1.3	Pixels required for a subframe selection query.	7
1.4	Example evolution of tile layouts in TASM.	8
1.5	Active learning manager overview.	11
2.1	Keyframes vs. delta frames in an encoded video.	15
2.2	Keyframes vs. delta frames in a tiled video.	16
2.3	Example tile layout.	17
2.4	Valid vs. invalid tile layouts.	18
2.5	Model inference and feature extraction for video inputs.	19
4.1	Examples of video partitioned into tiles.	37
4.2	Overview of how TASM integrates with a VDBMS.	40
4.3	Various ways to tile a frame.	42
4.4	Non-uniform tile layout around cars.	43
4.5	The impact of layout duration on the number of pixels decoded.	44
4.6	Query time improvement and quality of the fastest tile layouts.	55
4.7	Query time improvement for uniform tile layouts.	56
4.8	The effect of tile granularity on query time	58
4.9	The effect of sequence of tiles (SOT) duration on query time and storage cost.	59
4.10	Ratio of the number of pixels decoded with a non-uniform layout to the number decoded without tiles vs. performance improvement.	60
4.11	Cumulative decode and re-tiling time for various workloads.	63
4.12	Speedup achieved with TASM over the Visual Road object detection workload.	65
4.13	Specialized model preprocessing throughput	66
5.1	VOCALExplore architecture.	76

5.2 Average F1 and cumulative visible latency after 100 EXPLORE steps. 95
5.3 Macro F1 and S_{max} for various sampling methods. 97
5.4 Macro F1 score when using the VE-SAMPLE (CM) sampling method. 99
5.5 Feature selection progress for K20 showing upper and lower bounds. 100
5.6 Macro F1 score when performing feature selection. 101
5.7 Model quality and latency for VE-variants. 106
5.8 Macro F1 score when performing feature selection with noisy labels. 107

6.1 VOCALExplore's user interface. 110
6.2 Model inference and feature extraction for audio inputs. 117
6.3 Early vs. late fusion of multimodal inputs. 120
6.4 Multimodal model performance across steps. 124

LIST OF TABLES

Table Number Page

4.1 TASM API. 40
4.2 Datasets used to evaluate TASM 53
4.3 Tiled model accuracy 67

5.1 VOCALExplore API. 75
5.2 Datasets used to evaluate VOCALExplore 93
5.3 Features used by VOCALExplore. 94
5.4 Feature selection correctness. 99

6.1 Multimodal model performance after training on all videos. 123

LIST OF ALGORITHMS

1 Pseudocode for incrementally adjusting layouts 49

ACKNOWLEDGMENTS

I have many people to thank for helping me reach this milestone. I would first like to thank my advisor, Magda, for her patience and neverending optimism. I feel incredibly fortunate to have been able to learn from both her technical knowledge and leadership skills over the past few years. I also want to thank Dan Suciu for being there at every milestone of this journey; I have learned so much from his questions and feedback. I need to thank Ranjay Krishna for leading me out of machine learning purgatory. I would also like to thank Steve Mussmann for patiently teaching me about math and active learning, and Eunice Jun for being such a wonderful collaborator and friend. I am grateful to our collaborators, specifically Apryle Craig and Aaron Wirsing, for sharing their data and being generous with their time when explaining the specifics of their domain.

I feel very lucky to have been part of an incredible database group at UW. Thank you to Brandon for teaching me about videos and for sharing your advice and feedback. To Dong for being a great deskmate and listening to all of my ramblings and complaints. To Enhao for becoming my new video buddy. To Laurel for making the "silent" data science lab very much not silent. I am incredibly grateful to everyone else in the lab that I've been fortunate to work, lunch, and hike with: Leilani, Junran, Ameya, Anton, Kyle, Moe, Remy, Yihong, Jieyu, Ryan, Walter, Parmita, Cong, Amrita, and everyone else that I crossed paths with. I have learned so much from everyone!

I could not have completed this without the support of my family. Their unconditional encouragement made this journey possible, especially during the many challenging times when I was stressed and contemplating quitting. Thanks for rolling with all of the highs and lows of the past few years! Thank you also to Tiffany, Andy, and Joe for dragging me out on runs even when I thought I was too tired.

Finally, I want to thank all of the wonderful people I was able to meet throughout this Ph.D., whether that was through collaborations, workshops, conferences, internships, intramural sports, or outreach events. It has been incredibly motivating to learn from so many intelligent and passionate people.

DEDICATION

The only way the magic works is by
hard work. But hard work can be fun.
Jim Henson

To Mom and Dad.

Chapter 1

INTRODUCTION

Increasing numbers of scientific and engineering domains rely on video data, which is information dense and relatively easy to collect. Domain experts collect and analyze video content for varied applications of interest. For example, naturalists attach mobile cameras to wildlife to observe and analyze deer activity patterns and dietary preferences [45] or shark foraging behavior and the responses of prey [69]. Stationary wildlife cameras help scientists observe and analyze aggressive interactions between bird species at feeders [99]. In many other domains, as well, video data supports analysis including: traffic analytics, with camera feeds from busy city intersections providing support for installing crosswalks at locations where jaywalking is prevalent [15, 62]; retail analytics, where they help to analyze the effectiveness of displays based on customer traffic [134]; and sports analytics, where they assist in the analysis of team and player performance [140].

Such video analytics applications are made possible by recent commercial and technological trends. First, high-quality cameras are now relatively inexpensive and thus more widely deployed. Further, inexpensive storage in the cloud and on hard drives enables vast video data repositories. Finally, advances in machine learning (ML) and computer vision (CV) facilitate the automatic extraction of semantic content from videos; these models operate over the pixel data to identify objects or activities.

Though information-rich and easy to collect, videos remain difficult to analyze for three main reasons:

1. **Volume and information density:** videos are captured at frame rates ranging from tens to hundreds of frames per second, and each frame contains thousands to millions of pixels depending on its resolution. Despite state-of-the-art compression algorithms, videos easily consume gigabytes of storage, and efficiently storing, retrieving, and indexing data at such a large scale is challenging.

2. **Compressed format:** because videos are stored in compressed formats, decoding them to access their raw pixels adds latency to applications that operate over video content.

3. **Lack of structure:** unlike relational data, which takes the form of tables with well-defined column headers, video data is unstructured. Analysis must be performed over its pixels to generate insights. Typically ML models are applied to automate this analysis, but efficiently training and applying these models requires ML expertise that is not (and should not be) the primary focus of domain experts, and is tedious even for those with relevant expertise.

The increasing demand for video data analytics applications, combined with the preceding challenges associated with efficiently using video data, are giving rise to a resurgence of research and innovation in topics such as powerful libraries [25, 49, 106, 10] and video database management systems (VDBMSs) [81, 20, 113, 160]. We focus in this dissertation on the latter.

VDBMSs support users in storing, processing, and querying video data. Figure 1.1 shows the architecture of a generic database management system, which consists of components for storing data efficiently, optimizing and executing queries, and a data model and language facilitating user and application interaction with the data. Relational database management systems (RDBMSs) such as PostgreSQL [11] implement this stack to support users in efficiently querying text and numeric data. They store data in row- or column-oriented [13] formats,

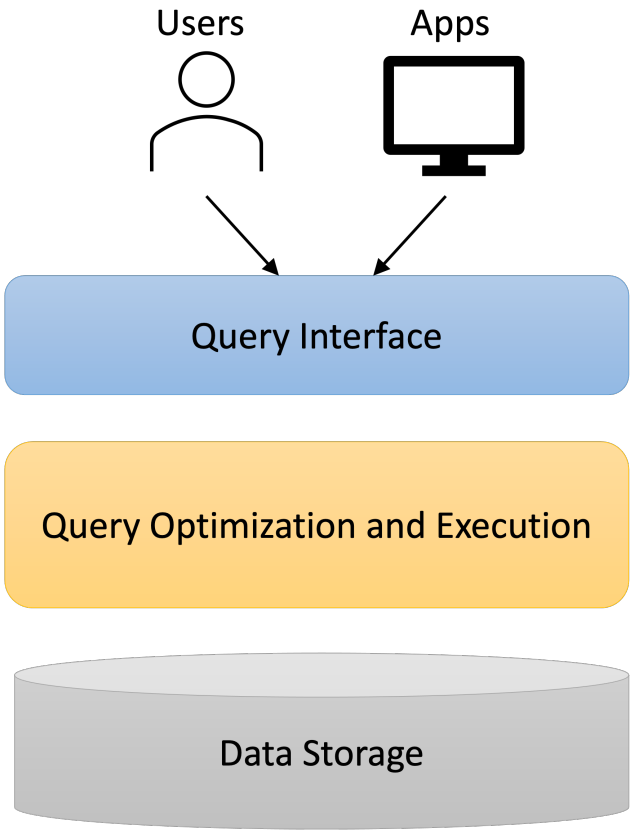


Figure 1.1: Architecture of a generic database management system (DBMS).

provide indexes [37] to accelerate data access, implement cost-based query optimizers [132] and efficient operators to accelerate query processing [115], and let users interact with data by specifying queries in the SQL query language.

VDBMSs reimagine this stack for video data. VDBMS storage managers must be able to operate efficiently over large quantities of compressed pixel data. Their query interface layer must let users express queries over the semantic concepts contained in videos, while the query execution layer must translate these semantic concepts into operations over pixels.

Existing VDBMSs [81, 82, 104, 160, 16, 20, 113] focus primarily on making query execution more efficient (we review related work in detail in Chapter 3). They assume access to a pretrained "oracle" ML model that accurately detects objects or activities of interest when

applied to video frames. Recent VDBMSs have developed techniques to more efficiently answer queries using this oracle model. They use faster, specialized models derived from the oracle to filter out irrelevant frames [104, 83] or directly answer queries [81, 16], perform content-based sampling of frames [113, 20], or store copies of the video at reduced resolutions that are faster to process but maintain model accuracy [67, 159, 86].

However, today's VDBMSs still fail to adequately support application needs, as exemplified by the following use-cases.

Example 1.1 (**Traffic monitoring**)**.** Consider an urban planner who needs to monitor traffic patterns in their city. They use traffic cameras located at intersections to count the number of vehicles, bicycles, and pedestrians to determine the busiest times of day. The planner is further interested in "close calls" between cars and cyclists [15], so they look specifically for instances with multiple cyclists in a turn lane concurrent with one or more cars. The planner must filter through the large quantity of traffic videos accumulated from each intersection over weeks to find these specific occurrences. Pretrained object detection models can filter out frames with no cars or bicycles; however, applying them over the entirety of the archived video data is slow and computationally expensive, particularly when query workloads operate over subregions of frames (e.g., just the pixels representing the intersection as in this example). Despite queries often requesting spatial subsets of frames, current VDBMSs decode and process entire frames due to intra-frame dependencies introduced during video encoding.

Example 1.2 (**Wildlife monitoring**)**.** A naturalist wants to understand the behavior and activity patterns of deer in the wild. They do so by attaching collars outfitted with cameras to wild deer [45]. Upon collecting the cameras after several weeks, they need to analyze the video data to answer their research questions. However, no off-the-shelf pretrained model can process and extract meaningful data from these videos: it is a unique domain, and the videos are captured from unusual angles. Therefore, the naturalist must manually label a sample of the videos and perform analyses over these labeled samples. This process is laborious and fails to use most of the captured data.

As highlighted in these examples, extracting insights from video content presents challenges both in the efficiency of query execution and in supporting domains that lack pretrained models to extract semantically meaningful information from their videos.

This thesis proposes techniques that address these two key unresolved or unaddressed challenges in current VDBMSs. First, *data loading*—which consists of reading raw data from disk and decoding it in order to answer queries—poses a bottleneck for applying ML models [86, 112] over videos. This limits query execution performance of VDBMSs because queries rely on executing one or more ML models over video frames (or parts of video frames). To execute those models, VDBMSs must first read the data from disk and decode it into raw pixels. Second, VDBMSs *assume access to a model that can act as an oracle*, which does not exist for most specialized domains. Therefore, many users with large video collections but no domain-specific model cannot access the capabilities of VDBMSs to accelerate their analyses.

The remainder of this Introduction elaborates on these challenges. Section 1.1 introduces TASM, a storage manager designed to optimize video data storage to accelerate data processing. Section 1.2 introduces VOCALExplore, a system designed to support users in exploring and building domain-specific models for their video data. In Section 1.3, we highlight the contributions of this thesis, and we present the organization of the remaining chapters in Section 1.4.

1.1 Opportunities to optimize video data storage

The first research question this dissertation poses is how to build efficient data storage managers for VDBMSs. Video data processing consists of two phases: *data loading* and *query execution* (typically performing inference on video frames), shown in Figure 1.2. During data loading, video content is read from disk, frames are decoded, and pixels are preprocessed according to the model's specification; this typically involves downsampling the video frames to a lower resolution, cropping them, and normalizing pixel values. In the inference phase, the preprocessed pixels are passed through an ML model to generate predictions.

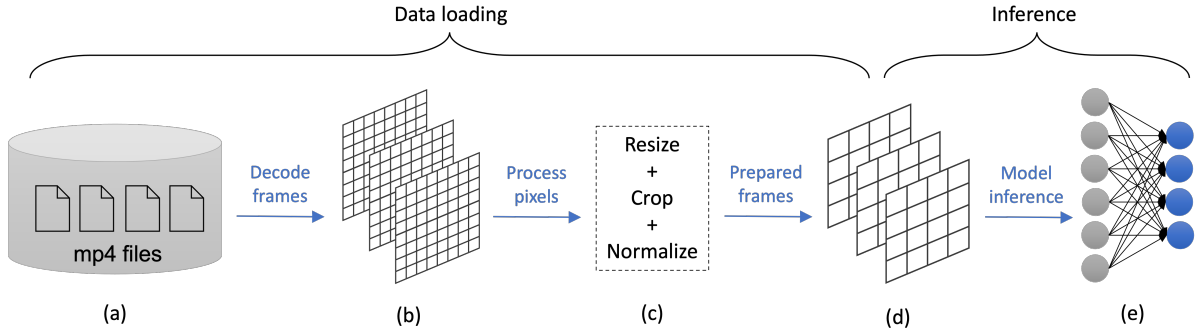


Figure 1.2: Video processing pipeline. (a) Videos stored in an encoded format on disk. (b) Decoded frames represented as matrices of pixels. (c) Preprocessing operations applied to pixels. (d) Processed frames represented as matrices of pixels. (e) Model inference performed on processed frames to generate predictions.

Considerable resources are available to accelerate inference work. GPUs and other specialized accelerators are optimized for the computations models perform, and deep learning frameworks [49, 106] provide high-level constructs to efficiently implement models. However, fewer resources address optimizing data loading, which often proves a bottleneck for ML pipelines [86, 112], and especially so for data sources like videos that are stored in a compressed format. VDBMSs require a powerful storage manager capable of optimizing the steps of loading data from disk and decoding the pixel information from the compressed format to enable efficient query execution.

We observe that a storage manager for video data should support queries that extract a subset of the data in time but also in space (i.e., spatial subregions of frames) For example, queries searching for frames containing cars can perform object detection only on frame regions showing the road (Figure 1.3a) since regions showing adjacent buildings or sky are not relevant. Or, an application that searches for a vehicle with a particular license plate number only needs to look at pixels corresponding to license plates. Finally, a query that collects examples of birds visiting a feeder to build a training set for a classification model can process only the pixels in frames corresponding to birds (shown in Figure 1.3b). These queries for pixels from a subset of the frame are called *subframe selection* queries.

(a) Subframe selection for cars in an image from the BDD dataset [163].

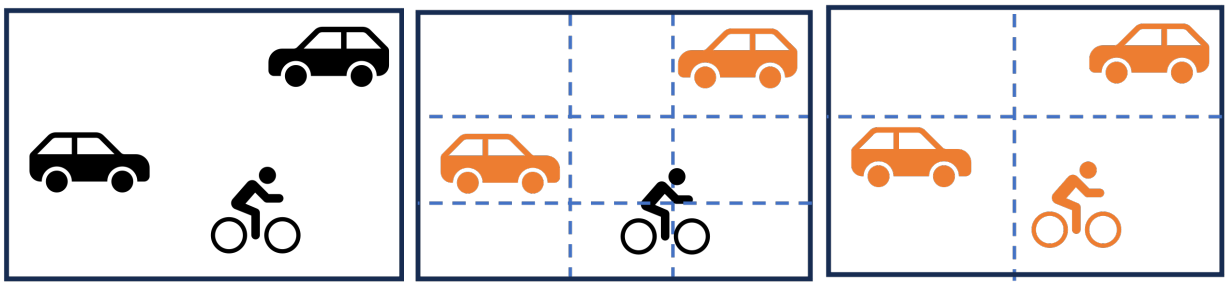
(b) Subframe selection for birds in a frame from the Netflix dataset [97].

Figure 1.3: Pixels required for a subframe selection queries. Only the non-frosted pixels will be used by the query, but storage managers currently decode the entire frame.

By default, the video encoding process creates spatial dependencies between different regions in a frame. Therefore, even if a query will process only a specific subregion, the entire frame still must be decoded. Further, VDBMSs store videos as a single encoded file or as a sequence of encoded files, split on time. This storage format provides temporal random access but no spatial random access: storage managers can jump to points in time that queries request, but they cannot selectively decode spatial subsets requested by queries. Today's VDBMSs thus fail to investigate storage optimizations that process only subregions of a frame.

To address this challenge, we design, implement, and evaluate TASM (Chapter 4), a storage manager for VDBMSs that *physically optimizes the layout of videos on disk and provides spatial random access to video frames*. It is designed to *accelerate subframe selection queries*, i.e., queries that operate over pixels from select regions within frames. TASM was introduced in ICDE'21 [41].

TASM's key contribution is splitting video frames into independently queryable tiles, thereby adding spatial random access to frames. It optimizes the tile layout (i.e., random access points) based on query workload and frame content. Notably, because the query



(a) Initial untiled video. (b) Tile layout around cars. (c) Tile layout around cars and bicycles.

Figure 1.4: Example evolution of tile layouts in TASM. (a) Initially, TASM has no information about video content and does not tile the video. Once it learns the location of some objects and sees queries for these objects, it adjusts the tile layout. (b) If TASM first encounters queries only for cars, it tiles around the cars. (c) If it then additionally encounters queries for bicycles, it adjusts the tile layout to accommodate both cars and bicycles. TASM does this for each group of frames.

workload and video contents are initially unknown, TASM dynamically evolves the layout for each group of frames over time as it observes queries and learns where objects are located in frames. Figure 1.4 shows the evolving tile layouts that TASM may traverse if it first encounters queries only for cars and then queries for bicycles, as well.

TASM can accelerate object detection queries in existing VDBMSs, particularly those that execute a fast model over a video to profile or filter frames [81], in what we call a *full scan phase*, before applying an expensive, high-quality object detection model to a select subset of frames. It cheaply profiles videos when they are initially ingested and designs an initial tile layout around *regions of interest* (ROIs). Initial tiling reduces preprocessing required for object detection queries because the VDBMS can perform its full scan phase over select tiles that contain ROIs rather than over entire frames.

We evaluate TASM on a variety of videos and workloads. We find that the tile layouts picked by TASM speed up subframe selection queries by an average of 51% and up to 94% while maintaining video quality. Further, (1) TASM's dynamic tile layout algorithm

automatically selects layouts that improve performance after only a small number of queries, and (2) TASM's ROI-based tiling accelerates the throughput of the full scan phase of object detection queries by up to 2× while maintaining high query accuracy.

TASM's code is available at `https://github.com/uwdb/TASM`.

1.2 Early video data exploration

Current VDBMSs are designed to support users in executing queries *when users know what they are looking for* and *have a model that can detect the objects and events they are interested in.* For example, users can apply an object detection model like Faster R-CNN [125] to find cars or bicycles in frames, or an activity classification model like MViT [50] to detect video segments where someone is riding a bike. However, despite being a critically important part of the data management lifecycle, early data exploration over video datasets has received scant research attention, particularly when no oracle model exists. During early exploration, users familiarize themselves with their dataset to learn the types of events they captured and what these events look like in the videos. As part of today's exploration process, users must manually search for a diverse set of examples of activities of interest to train a domain-specific model.

Manually searching through a large video dataset is tedious; it requires careful scrubbing through long videos or navigating thousands of individual files for collections of short videos. Further, users who collect large video datasets typically have domain expertise to understand the content captured by the videos but may lack the ML expertise to effectively train a domain-specific model. Even for users who have ML expertise, training a model over videos is non-trivial. Thus, users may be comfortable annotating the videos they collect but not using these annotations to train a model and then applying the model to automatically annotate the remaining unlabeled portion of their dataset.

Though the ML field has contributed many discrete techniques and tools to build high-quality models, combining them into an efficient end-to-end system is nontrivial. Specifically, a complete solution must support: (1) *sample selection* (i.e., which examples the user should

label to maximize model performance if they have a limited budget), (2) *feature representation extraction* (i.e., generating feature vectors from video frames that models can use), and (3) *model training and inference*. Implementing and efficiently coordinating these components is difficult due to the volume and storage format of videos; further, decoding and preprocessing videos to extract feature representations is a bottleneck that makes it impractical to fully index large datasets during preprocessing.

Additionally, no guidance is available about which combination of techniques would be optimal for an arbitrary dataset. The best acquisition function (i.e., sample selection method) depends on the dataset, and random sampling often performs at least as well as more sophisticated techniques [87]. Similarly, the most effective feature representations are dataset-dependent (as we show in Chapter 5). Without systematic support to navigate these decisions, users have no way to know which techniques to implement for a specific task.

The second research question this dissertation poses is how to address the challenge of supporting early data exploration, especially when no off-the-shelf oracle model exists. To this end, we introduce VOCALExplore, designed as *an interactive system to support users in exploring their video data and building domain-specific models to use in downstream analysis tasks*. VOCALExplore is a component of the larger VOCAL (Video Organization and Interactive Compositional AnaLytics) project [42]. Users of VOCALExplore iteratively view video segments, either ones they select or those automatically selected by the system, and apply labels to them. VOCALExplore uses these labels to train domain-specific models by integrating a variety of ML techniques to support end-to-end video data exploration and model building—from video sampling to feature extraction to training models on video data. It requires no ML knowledge or hyperparameter tuning from the user. Importantly, it also does not demand expensive data preparation or preprocessing; instead, it is designed as a "pay-as-you-go" system where users can *immediately* begin interacting with their data, and the system improves model quality as the user spends more time labeling their data. VOCALExplore will appear at VLDB'24 [43].

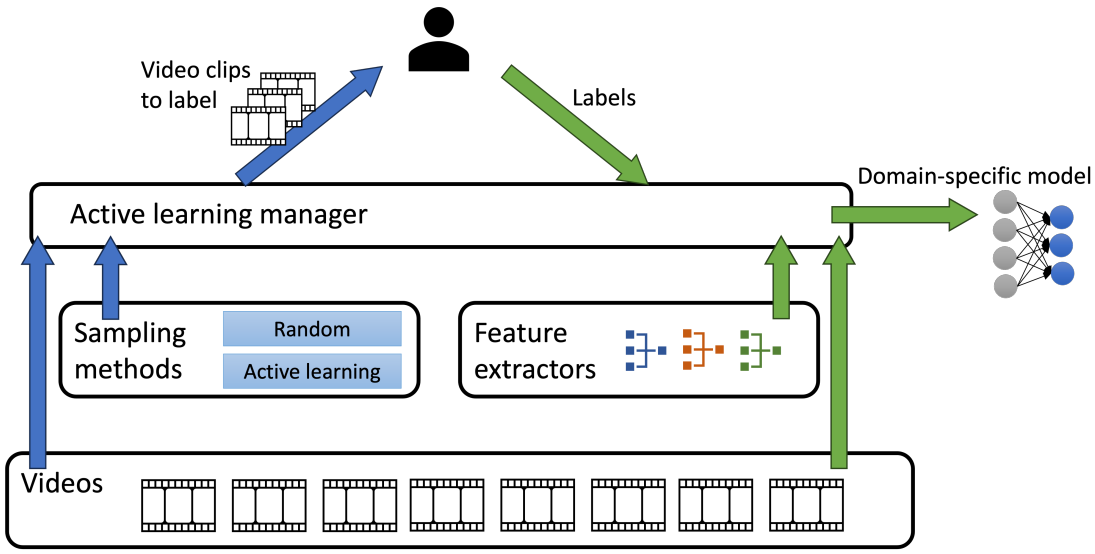


Figure 1.5: Active learning manager (ALM) overview. As shown on the left with the blue arrows, the ALM uses the dataset's characteristics to pick a sampling method. It uses this sampling method to select video clips for the user to label. As shown on the right with the green arrows, the ALM uses these labels to train a domain-specific model on top of features extracted from the video clips. The ALM automatically picks a feature extractor whose features perform well on the given domain.

VOCALExplore uses an *Active Learning Manager* (ALM) to produce high-quality models. The ALM first samples video segments for the user to label, as shown in Figure 1.5. Many candidate acquisition functions have been proposed by the ML community, but the best choice depends on the dataset, and random sampling often performs at least as well as more expensive functions [87]. It dynamically selects an appropriate acquisition function based on observed skew in the collected labels.

The ALM next selects an appropriate feature representation for the video segments to train models and perform inference. Though it is an accepted technique to extract feature embeddings from models pretrained on other tasks [24, 59, 51], no guidance exists on how to select the best pretrained model to use as a feature extractor. We show that the quality of the

model trained over collected labels largely depends on choosing a suitable feature extractor. The ALM dynamically selects the best feature extractor for each dataset using an algorithm based on rising bandits [96].

VOCALExplore uses a *Task Scheduler* (TS) to reduce the latency associated with producing high-quality models. The TS executes low-priority tasks in the background to avoid blocking responses to API calls. It dynamically schedules model training tasks based on observed user labeling and model training latencies to produce up-to-date models that are ready for inference upon the next user interaction.

We evaluate VOCALExplore on both a domain-specific deer dataset [45] and on a standard activity classification video dataset [139]. We show that the system produces models that match the quality of the best combination of acquisition function and feature extractor with (1) zero preprocessing latency, and (2) a user-visible latency of less than one second per labeling iteration.

VOCALExplore's code is available at `https://github.com/uwdb/VOCALExplore`.

1.3 Summary of thesis contributions

This thesis introduces techniques to address the two limitations highlighted previously: (1) limited storage-level optimizations despite data access being a bottleneck for query execution, and (2) lack of support for early data exploration and domain-specific model development. Specifically, our core contribution is two systems that feature new techniques for solving each challenge. The first system, TASM, is a low-level storage manager for video data that provides new opportunities for spatial random access to video frames. The second system, VOCALExplore, is a high-level system that supports users in exploring their data and building domain-specific models useful for further analysis. In sum, this thesis contributes novel approaches to video storage and exploration that make VDBMSs more efficient to run and more useful for video datasets from diverse domains.

1.4 Thesis Organization

Chapter 2 presents background information relevant to the work presented in this dissertation, and we discuss related work in Chapter 3. Chapter 4 introduces TASM, and Chapter 5 introduces VOCALExplore. We describe in Chapter 6 the system design, interface, and extensions to VOCALExplore. Finally, we provide concluding remarks in Chapter 7.

Chapter 2

BACKGROUND INFORMATION

This chapter highlights concepts and terminology referenced by later chapters.

2.1 Video encoding and storage

This section, which reviews how videos are encoded and introduces the concept of using tiles to encode videos, is not intended to be comprehensive, but rather to provide sufficient background for subsequent chapters. For a more detailed discussion, we refer the reader to [142, 111].

Due to their large size, videos are stored as encoded files. Video codecs—such as H264 [1], HEVC [3], and AV1 [44]—specify encoding and decoding algorithms for (de)compressing videos. Though their specific algorithms differ, their high-level approach is the same, as we now describe.

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 other GOPs and is typically one second in duration, but it can be configured to other values. The first GOP frame, called a *keyframe*, is encoded with no dependencies on other frames. Remaining frames in a GOP are encoded as *deltas* from surrounding frames.

Because video decoding can begin at any keyframe, keyframes let GOPs act as temporal random access points into the video. To retrieve a specific frame, the decoder begins at the closest keyframe preceding the frame being retrieved, as shown in Figure 2.1. Keyframes require large amounts of storage because they use a less efficient form of compression than

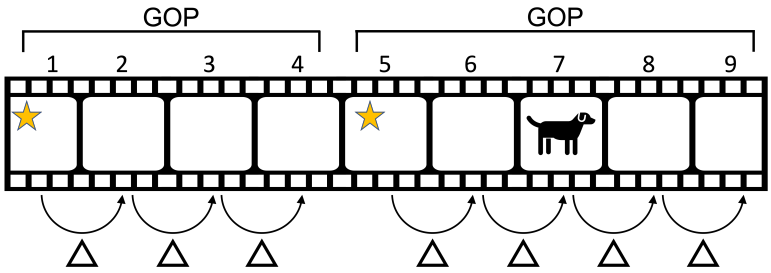


Figure 2.1: Keyframes vs. delta frames in an encoded video, simplified for ease of presentation. Starred frames 1 and 5 are keyframes and act as temporal random access points. Remaining frames are delta encoded. To access the frame with the dog (frame 7), frames 5, 6, and 7 must be decoded.

delta frames, so the number of keyframes impacts a video's overall storage size. Videos stored with long GOPs have a smaller storage footprint than those stored with short GOPs, but they also have fewer random access opportunities.

Tiles. Compressed videos do not generally support the decoding of spatial subregions 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 only a small region is needed. However, modern codecs provide a feature called *tiles* that enables splitting of frames into independently decodable regions. Figure 2.2 illustrates this concept.

Like frames, tiles are also 2D arrays of pixels. However, a tile contains only the pixels for a rectangular portion of the frame; the full frame is recovered by combining tiles. Tiles introduce spatial random access points for decoding, though temporal random access is still provided by keyframes. To decode a region within a frame, only the tiles that contain the requested region are processed. This flexibility involves tradeoffs in quality: tiling can produce artifacts at tile boundaries [141], reducing the visual quality of the video. As such, careful selection of tile layouts is critical for high-quality query results. Tiles are applied to all frames in a GOP, so decoding a tile in a delta frame requires decoding that tile in all frames starting from the preceding keyframe, as shown in Figure 2.2.

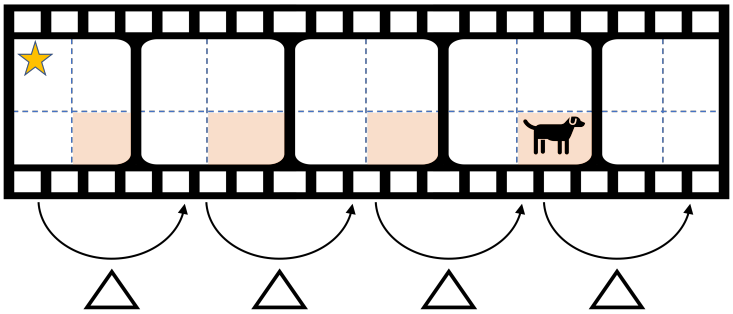


Figure 2.2: Keyframes vs. delta frames in a tiled video. All frames in this GOP are encoded with the same 2×2 tile layout, marked by the dashed lines. The first frame is a keyframe, and the remaining frames are delta-encoded. To decode the tile with the dog, the lower-right tile must be decoded in all preceding frames, starting from the keyframe, as illustrated with the shaded tiles.

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; Figure 2.3 shows a sample tile layout. 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 where a single tile encompasses the entire frame: $\omega = (1, 1, \{frame\_height\}, \{frame\_width\})$.

Valid layouts [3] require tiles to be partitioned along a regular grid, meaning that rows and columns must extend through the entire frame, as shown in Figure 2.4. 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 to the tile layout must take place at GOP boundaries, so every new layout must start at a keyframe. Therefore, changing the tile layout incurs high storage overhead for the same reason that it does for starting a new GOP. The cost of executing a query over video encoded with tiles is proportional to the number of pixels and tiles that are decoded.

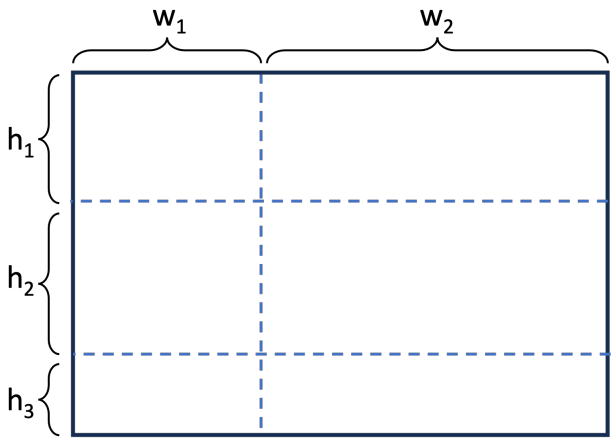


Figure 2.3: An example tile layout $L = (3, 2, \{h_1, h_2, h_3\}, \{w_1, w_2\})$ with 3 rows and 2 columns.

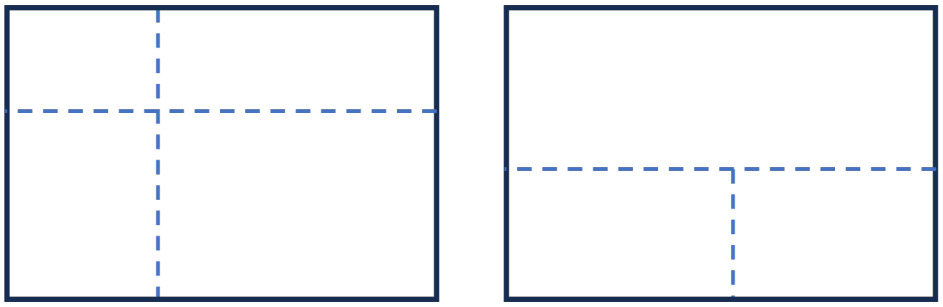
Stitching. Tiles can be stored separately, but they must be combined to recover the original video. Tiles can be combined without an intermediate decode step using a process called *homomorphic stitching* [65], which interleaves encoded data from each tile and adds header information so the decoder knows how to arrange the tiles.

2.2 Object and activity recognition in videos

We now provide background information on how feature embeddings are extracted from videos and used to train object and activity detection models. As in Section 2.1, our intent is to provide background for subsequent chapters, not a comprehensive treatment of the topic. We refer the reader to [59, 144, 129, 171] for further details.

2.2.1 Video models

To extract semantic information from videos, a common approach today uses a deep learning (DL) model. Different types of models exist. An *image model* takes as input an image or video frame and produces predictions for objects or activities in that frame. A *video model* does the same for a sequence of frames (i.e., a video clip), as shown in Figure 2.5a.

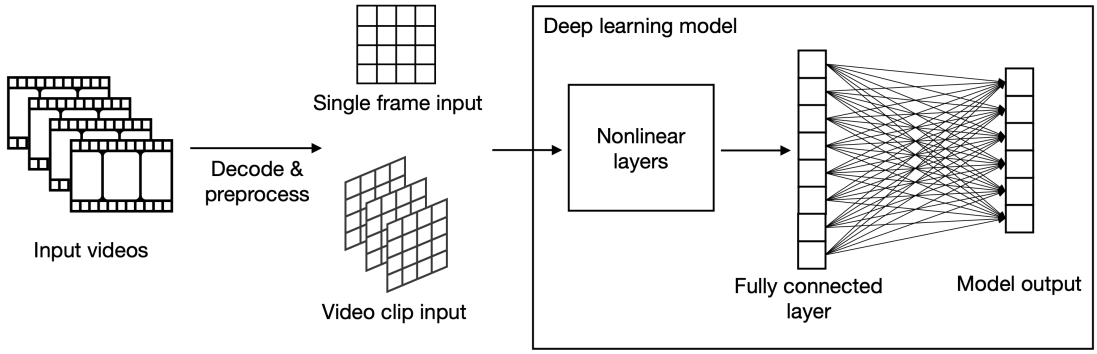


(a) Valid layout. (b) Invalid layout.

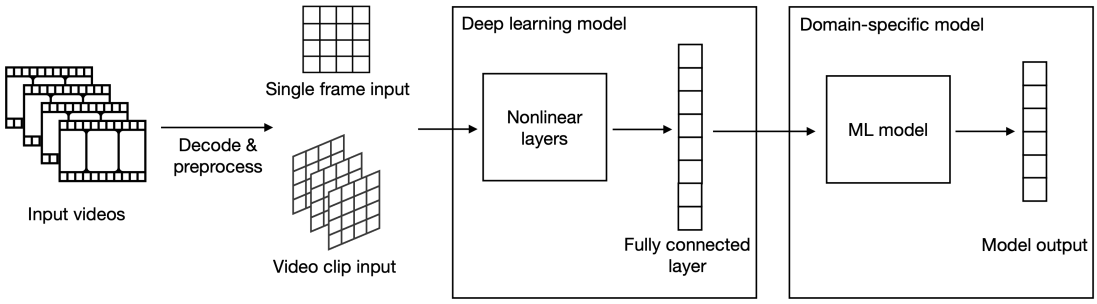
Figure 2.4: Valid vs. invalid tile layouts. (a) This is a valid layout because tile boundaries (marked by dashed lines) extend through the entire frame. (b) This is not valid because the vertical boundary does not extend through the entire frame.

End-to-end models operate over raw pixels, typically after downsampling, cropping, and normalizing the pixel values in frames. Modern DL model architectures are constructed as a sequence of stages (commonly convolutional or transformer-based) that implement nonlinear transformations over the inputs. Final predictions are generally produced using a fully connected layer.

End-to-end training of DL models requires large numbers of labeled examples, typically on the order of millions [46]. Generating sufficient data to train models from scratch for new domains is therefore infeasible, so a common technique in computer vision involves *transfer learning* [24, 59]. Transfer learning starts with a model that was pretrained on a large dataset that is readily available and known to produce reliable results [51]. For example, ImageNet [46] is a common pretraining dataset for object classification, and Kinetics400 [90] is frequently used to pretrain activity classification models. Applying transfer learning means there is no need to start training using a model initialized with random weights; instead, transfer learning fine-tunes model weights using examples from the target domain and normal backpropagation computations. Depending on the application, all layers, or just a subset of them, can be tuned, and the weights of remaining layers are frozen.



(a) End-to-end model pipeline.



(b) Using a pretrained model as a feature extractor. The domain-specific ML model can have any architecture.

Figure 2.5: Model inference and feature extraction for video inputs. Image models operate over individual frames, while video models operate over frame sequences.

Another use-case for transfer learning leverages pretrained models as feature extractors and then trains a model from scratch on top of these features, as shown in Figure 2.5b. A *feature extractor* takes as input one or more video frames and produces as output a *feature vector*, also called an *embedding*, which is a d-dimensional vector in $\mathbb{R}^d$, where the dimension d may vary across feature extractors. This workflow assumes that features from a pretrained model capture general knowledge that will be useful in a new domain. A common technique extracts feature representations from the final fully connected layer of a pretrained model, though any layer can be used to generate embeddings. Features extracted from the pretrained model can be materialized to avoid repeatedly performing expensive decode and transform operations over the encoded videos.

2.2.2 Other feature representations

Though Chapter 5 considers using only pretrained models as feature extractors, alternative feature representations are possible.

Basic features. Frameworks such as OpenCV [25] support extraction of basic features from raw pixels, such as optical flow or SIFT descriptors [102]. Historically, basic features have been manually designed and tuned for specific use-cases, such as action recognition [147] or duplicate video detection [136]. Though they are often simple to compute compared to performing inference over DL models, extracting basic features on CPUs is not necessarily more efficient than extracting pretrained embeddings on GPUs due to advancements in accelerators and optimizations in DL frameworks.

Self-supervised features. *Self-supervised training* produces embeddings specific to a given dataset by training a DL model over unlabeled videos [129]. It uses objectives that rely on the similarity or dissimilarity of inputs that can be known even without labels (e.g., embeddings from different clips of the same video should be similar [52]). This approach is expensive since it requires training a large model and repeatedly extracting updated embeddings from videos. However, self-supervised features are the most domain-specific because the embedding is learned from the unlabeled dataset itself.

2.3 Summary

This chapter introduced background information and terminology on video encoding and decoding, specifically using tiles (Section 2.1), and on the use of ML models to automatically extract semantic information from videos (Section 2.2). The rest of this dissertation builds on these ideas for the design of a tile-based storage manager (Chapter 4) and a system that uses pretrained image and video models to help users quickly build domain-specific ML models for their video datasets (Chapter 5).

Chapter 3

RELATED WORK

Video data management is a broad area with important work tackling each of the components of such a system from data models and query languages to query execution, query optimization, and storage management. In this section, we give a summary of the relevant literature. We start with a brief overview of earlier, seminal work (Section 3.1). We then dive into recent work on query optimization, query execution, and storage management (Section 3.2). We end by presenting complementary work to TASM (Section 3.3) and VOCALExplore (Section 3.4).

3.1 Early work on VDBMSs

Video data management has long been an area of interest in the data management community. The seminal systems described in this section did not have access to the machine learning and computer vision capabilities developed in the past few years, and therefore they were limited to analyzing videos based on attributes derived from the pixels (e.g., motion and color) or descriptions manually specified by users.

VideoQ [30] is an online search engine that supports object-based indexing and spatiotemporal queries. It introduces the idea of formulating a query over video using an animated sketch, where each object in the sketch is associated with motion and temporal duration. VideoQ indexes videos using features derived from color, texture, motion, and shapes in the video frames.

OVID [118] is an object-oriented VDBMS designed around the data model of a *video-object*, which is a sequence of frames described with a number of key-value attributes. OVID introduces operations for deriving new video-objects from existing ones. Its query language, VideoSQL, operates over the key-value attributes of the video objects.

BilVideo [48] supports spatio-temporal queries. It stores facts about trajectories and spatiotemporal relationships between video objects in a knowledge-base. It also stores pixel-based metadata such as color, shape, and texture in a feature database. Users can visually specify queries by sketching the desired locations of objects or the trajectory of one or more objects.

VideoAnywhere [137] provides a framework for supporting video search over many different sources of videos, such as the web, TV, and DVDs. Providing semantic interoperability is an important component of VideoAnywhere, which is achieved by mapping XML tags used by the video sources to a unified ontology that the user can query over. Delaunay^{MM} [39] similarly supports searching over possibly many sources of multimedia and associated metadata, however its focus is on presenting an integrated view of the information to the user. It does so via a *virtual document*, where each element on the page is associated with one or more queries to an underlying data source.

3.2 Recent work on VDBMSs

In this section we present recent work on VDBMSs. In contrast to the systems described in the previous section, the VDBMSs described here benefit from the capabilities of machine learning (ML) models to automatically extract high-quality semantic information from videos. As such, these new systems do not have to approximate semantic content using features like histograms derived from pixel values; instead, they directly extract and operate over the outputs of ML models. This paradigm shift creates new challenges of efficiently applying one or more models over large quantities of videos.

This section describes recent work in query optimization (Section 3.2.1), query execution (Section 3.2.2), data storage (Section 3.2.3), and other topics of interest (Section 3.2.4).

3.2.1 Query optimization

Query execution in current VDBMSs frequently requires executing and operating over the output of one or more ML models. Despite advances in accelerators and ML frameworks, applying models with many parameters to a large quantity of video frames can cause significant latency during query execution. To address this problem, the database community has investigated many techniques to reduce this latency. In this subsection we describe three such query optimization techniques: (1) using faster specialized models; (2) navigating between multiple models that have varying throughput and accuracy; and (3) task-specific optimizations.

Specialized models. We first discuss systems that use specialized models to accelerate queries. The key observation that motivates incorporating specialized models into query execution is that while a large, general-purpose model can perform well on many tasks, the tasks for individual queries are often much narrower. For example, an object detection model like Faster R-CNN [125] can localize and identify tens to hundreds of object classes, whereas a query asking for video clips with cars only needs to solve a binary problem of detecting whether or not each frame contains a car. A much smaller, and therefore faster, model can be used to perform the simpler task. In this section, we refer to the high-quality, general model as the "oracle" model.

NoScope [82] uses specialized models to accelerate queries that find frames containing particular objects. It trains the specialized model using outputs from the oracle model, and it uses the specialized model to find frames that are highly (un)likely to contain the target object. It only applies the expensive oracle model on frames where the specialized model is uncertain. Similarly, Probabilistic Predicates [104] (PP) uses specialized models to filter out irrelevant frames when executing queries that search for frames satisfying some conditions. PP's query optimizer supports queries with complex predicates by deciding how to apply and combine multiple models to efficiently filter query inputs while maintaining query accuracy.

BlazeIt [81] and ABae [84] use specialized models to accelerate aggregation queries while providing statistical guarantees on the correctness of the results. BlazeIt trains a specialized model that outputs the number of objects in each frame. It uses this specialized model directly to answer the query if its accuracy is high enough. Otherwise, it uses this model as a control variate to reduce the number of frames that must be sampled and processed with the oracle model. ABae instead uses the output of a specialized model to group video frames. It performs stratified sampling over these groups and applies the oracle model to the sampled frames to compute an approximate aggregate.

Tahoma [16] trains a *cascaded* model instead of a collection of specialized models. The complexity of each level in the cascaded model increases. Tahoma reduces query latency by stopping the inference process early when the desired accuracy is met.

Multiple models. Second, we discuss systems that utilize a collection of models to optimize queries. Typically the various models have different cost/quality tradeoffs, which enables more opportunities for query optimization.

FiGO [28] supports queries to find frames containing target objects. It splits videos into variable-sized segments and picks a model to apply to each segment. FiGO only uses slower but more accurate models on challenging segments, which are identified as such by looking at label agreement across models.

EVA [160] uses a semantics-based reuse algorithm to perform model selection. For each logical task (e.g., object detection), it has many candidate models. It considers both model accuracies and the results materialized by previous queries when selecting a model to use. It further optimizes queries by reordering model-based predicates based on their execution cost.

Rather than assume knowledge about the semantic relationships between models as in EVA, VIVA [126] instead optimizes queries using relational hints, which are specified by the user to give the optimizer more information about the relationships between black-box models. For example, *replacement hints* specify that one model may replace another model,

and *filtering hints* specify that a fast model can act as a filter for an expensive model. VIVA introduces a hint-aware query optimizer that selects the optimal sequence of models to execute a query while meeting accuracy constraints.

Task-specific query optimization. Finally, we discuss systems that implement optimizations specific to tasks such as object tracking and action localization.

Miris [20] and OTIF [21] accelerate object tracking queries. Miris computes trajectories by resolving object detections across frames, but it minimizes the number of calls to the object detector by varying the framerate that is used for sampling. It initially uses a low framerate and then only samples more frames from video segments where it is uncertain about a trajectory. Rather than compute trajectories one-by-one, OTIF instead designs an efficient technique to preprocess an entire video and extract *all* tracks. It then performs trajectory queries on top of the extracted track information.

Zeus [35] accelerates action localization queries using reinforcement learning (RL) to dynamically adjust the resolution and framerate used when applying an action classification model. It only uses more expensive and accurate high-resolution frames and fine-grained sampling for segments that are expected to contain the target action. Zeus additionally uses the RL agent to dynamically adjust the duration of each video segment.

3.2.2 Query execution

Executing queries over large video datasets is resource-intensive due to the large volume of frames that must be processed, especially for datasets collected over long durations of time. Systems such as Scanner [121], SVE [75], and Optasia [103] efficiently execute queries over large quantities of videos by parallelizing processing. They represent query plans as dataflow graphs and distribute the plan execution over clusters of machines. Scanner is a general-purpose execution engine that is capable of distributing operations over heterogenous hardware. Optasia is specifically designed for surveillance-type analysis on top of videos, and

it operates over the SCOPE [29] engine. Finally, SVE parallelizes processing both across groups of frames, and also across tracks within a group of frames (i.e., it processes audio and video tracks in parallel).

3.2.3 Data storage

Storage managers in VDBMSs organize videos as encoded files due to their large size. From a storage footprint standpoint, it is resource-prohibitive to directly store videos as raw pixels or individual frames. However, videos must be decoded to a pixel representation before applying ML models.

Multiple formats. The key idea of storing multiple versions of a video segment is to enable throughput/accuracy tradeoffs within queries. It is faster to decode a low-resolution video, however, the accuracy of models run on top of degraded video may suffer. VStore [159] profiles downstream operators to generate accuracy curves for various encoding settings. It then encodes videos in multiple formats. Given a query, it picks the physical representation with the fastest preprocessing speed that still meets the query's accuracy specification.

Smokescreen [67] stores degraded videos after profiling tradeoffs between accuracy and user-defined objectives (e.g., maximum resolution) for various destructive interventions on videos (e.g., decreasing resolution or sampling rate). It does not require access to the full-resolution video. Instead, it computes error bounds from query results produced by operating over degraded videos.

VSS [64] caches multiple versions of a video segment based on the query workload (e.g., raw pixels if a previous query requested RGB values). It combines and transforms these cached versions to quickly produce video in formats requested by a new query. It also automatically identifies and eliminates redundancies introduced by cameras with overlapping fields of view by jointly compressing the overlapping region.

While Smol [86] is a runtime engine rather than a storage manager, it explicitly considers the input format of videos as a parameter when training models. It optimizes preprocessing speed by using low-resolution frames that are fast to decode and by utilizing partial decoding whenever the codec supports it. Its optimizer also reorders preprocessing operators and distributes them across CPUs and GPUs to maximize preprocessing throughput.

Vignette [108] minimizes the storage size of videos while maintaining perceived quality by splitting frames into tiles and optimizing the encoding parameters for each tile based on its content. Tiles that contain content that is not salient, i.e., viewers do not focus their perceptual attention on that region, are encoded with low resolution. Vignette only considers uniform tile layouts (i.e., each tile has the same size).

Indexing. VDBMSs implement indexing over feature representations of videos to accelerate queries for similar frames as well as selection and aggregation queries. Video-zilla [74] clusters feature vectors to identify representative frames or scenes. TASTI [85] also clusters feature vectors and only applies expensive operations, such as object detection, on frames represented by the centroids. It propagates the results to frames in the corresponding clusters. ADAM-pro [58] builds indexes using Locality-Sensitive Hashing [78], Spectral Hashing [153], and Vector Approximation-Files [152] to support efficient nearest neighbor searches over feature representations. Another line of research focuses on building indexes over precomputed object detections and trajectories to efficiently execute spatio-temporal queries [32, 34, 33]

3.2.4 Miscellaneous

This section introduces related work that does not neatly fit into the previously-discussed categories.

Data drift. VDBMSs rely on deep learning models to extract information from videos. However, the accuracy of these models may drop when lighting conditions change, or, in the case of a non-stationary camera, when moving into a novel environment. ODIN [143] automatically detects and copes with data drift. ODIN detects drift using an adversarial autoencoder to learn the distribution of seen frames. It then compares the distribution of

current data to previously-seen frames. To handle data drift, ODIN clusters frames, and each cluster has an associated specialized model that performs well on those data frames. If newly-seen frames form a new cluster and do not yet have a trained specialized model, ODIN uses an ensemble of models from nearby clusters.

Panorama [170] targets the problem of expanding query vocabularies beyond the labels a model is originally trained to detect without an expensive re-training step. Panorama starts with an oracle model and uses its outputs to train PanoramaNet, which is a model architecture that generates embeddings for frames. Panorama supports queries for new vocabulary entities by performing nearest neighbor search over the embeddings. Two embeddings are said to represent the same object if their difference is below some threshold. Panorama supports verification (i.e., are two objects the same) and recognition queries.

Benchmarking. Unlike relational DBMSs, there are few standardized benchmarks for comparing the runtime performance and accuracy of VDBMSs. Visual Road [66] is a recently proposed benchmark that is designed to measure the runtime performance of common video manipulation tasks, such as cropping, resizing, and applying models. Evaluating the correctness of queries over videos is expensive due to the cost of applying high-quality oracle models on each frame. Visual Road works around this problem by using synthetic videos generated from a video game engine where the ground truth (i.e., objects contained in each frame) is known.

Multimedia retrieval. The multimedia community conducts annual competitions to measure the efficiency of exploratory video retrieval tools [60, 100]. These competitions focus on searching for known items in a video or retrieving relevant video segments for an ad-hoc query from a collection of videos. For example, with known item search, the task is to take a video segment, visual sketch, or keywords as inputs and return similar video segments as output. Vitrivr [128] supports this task by performing nearest neighbor search over various types of features extracted from video frames, however users must specify how to weight

the different features. It includes a storage engine optimized for queries over both features and metadata [55], a search engine to efficiently evaluate queries [127], and a web-based user interface [72].

More recently, systems like Vitrivr-Explore [70] and SOM-Hunter [94] incorporate user relevance feedback that classifies video segment outputs as relevant or not. This feedback is used to train self-organizing maps [91] to more effectively find segments similar to the ones that the user marked as relevant.

3.3 Related work on video data storage

This section explores systems and techniques related to TASM, which will be discussed in more detail in Chapter 4. Recall from Section 1.1 that TASM is a storage manager for VDBMSs that optimizes video storage and retrieval for query workloads that process subregions of frames. For example, if queries request just the portion of frames showing cars or pedestrians, TASM reduces the number of unrelated pixels that must be decoded and processed. TASM does so by splitting frames into independently-decodable tiles, and by selecting the layout of these tiles based on the observed query workload.

Incremental indexing. TASM's incremental tiling approach is inspired by database cracking [77, 61], which incrementally reorganizes the data processed by each query, and online indexing [27], 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 [40]. TASM extends these relational storage techniques to provide efficient access to video data.

Spatial indexing in videos. Other application domains have observed the usefulness of retrieving spatial subsets of videos. The MPEG DASH SRD standard [116] 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.

Video storage systems. As described in Section 3.2.3, there has been significant research around designing storage managers for VDBMSs. Storage managers thus far have primarily investigated how to accelerate query processing by storing multiple representations of a video to enable accuracy/throughput tradeoffs, rather than by reducing the amount of extraneous pixels that are decoded and processed in the course of executing queries.

VStore [159] encodes videos with multiple encoding settings after profiling downstream operators. When processing a query, it picks the version with the fastest processing speed that meets the query's accuracy specification. It improves query performance by reducing the quality of the video to make it faster to process, whereas TASM maintains video quality and improves performance by processing fewer pixels. Additionally, VStore must profile all downstream operators to determine the encoding parameters, while TASM can work incrementally as queries are processed.

Smol [86] trains models over reduced-fidelity frames, which are faster to decode, to accelerate training and inference. It jointly optimizes video resolution and model architecture to achieve a desirable accuracy/throughput tradeoff. However, like VStore, Smol only considers optimizations that reduce the quality of videos. Additionally, while Smol supports partial decoding to efficiently retrieve regions of interest, it does so only for images. In contrast, TASM re-encodes videos with tiles to enable partial decoding even if the original input video did not support it.

Vignette [108] is a storage manager optimized for perceptual compression, meaning that it reduces the quality of stored videos to minimize storage size, but does so in a way that minimizes the *perceived* quality loss. Like TASM, Vignette encodes videos with tiles. However, Vignette optimizes the tile layout based on saliency, which indicates the regions where users focus their attention, whereas TASM optimizes the tile layout based on the pixels retrieved by an observed query workload. Further, while TASM considers flexible tile layouts based on the locations of objects, Vignette only considers a fixed set of uniform tile layouts (e.g., 5×10) where all tiles have the same dimensions.

General query processing systems. If we consider more general VDBMSs, TASM could be incorporated into the systems described in Section 3.2.1 that optimize the process of extracting semantic content from videos during query processing. For example, BlazeIt [81] and NoScope [82] apply specialized neural networks that run faster than general models. They could use TASM to retrieve regions of interest from frames as input to these specialized models, rather than retrieving entire frames. TASM could also be applied to systems that filter frames before applying expensive models, such as probabilistic predicates [104] and ExSample [113] which use statistical techniques; MIRIS [20] which uses sampling; and SVQ [156] and *IC* and *OD* filters [93] that use deep learning filters. The bottleneck for these inexpensive filtering steps is generally decoding and preprocessing frames [86]. As we discuss in Section 4.2.4, TASM can tile around regions of interest and only decode these tiles as input to the filter to reduce the preprocessing overhead.

VDBMSs that focus on efficient end-to-end query execution could incorporate TASM as their storage manager to further accelerate performance because TASM's optimizations are complementary to their existing optimizations. These are systems such as LightDB [65], Optasia [103], and Scanner [121] which accelerate queries through parallelization and deduplication of work, and VideoEdge [76] which distributes processing over clusters. Finally, Panorama [170] and Rekall [53] expand the set of queries that can be executed over videos, which is orthogonal to video storage.

3.4 Related work on video data exploration

This section explores systems and techniques related to VOCALExplore, which will be discussed in more detail in Chapter 5. Recall from Section 1.2 that VOCALExplore is an interactive system that supports users in exploring their video datasets and building domain-specific models over their videos. It is designed for datasets where no off-the-shelf pretrained model exists to automatically extract semantic content from the videos. Users can browse their videos either via a manual search, or by using VOCALExplore to automatically select

videos that are expected to improve the domain-specific model when labeled. VOCALExplore incorporates new labels to train updated models, as well as to pick a feature extractor that works well for the given dataset.

Video querying systems. As discussed in Section 3.2.1, there are many recent examples of video querying systems that support executing pretrained models as UDFs over videos such as EVA [160], VIVA [126], and others [81, 33, 104, 28, 16]. However, in contrast with VOCALExplore, these systems assume access to a pretrained model. Rather than guiding the user through an exploration and labeling process, they focus on efficient execution of queries over the outputs of pretrained models.

Like VOCALExplore, Panorama [170] does not assume a fixed vocabulary of classes. However, rather than train a model capable of predicting new classes, it supports queries over novel labels using embedding similarity. While VOCALExplore also uses embeddings as input to its domain-specific models, Panorama requires an oracle model to train its embedding network. VOCALExplore avoids the need for an oracle model by extracting embeddings from models pretrained on other video datasets. Finally, Panorama focuses on recognition and verification queries rather than exploration and domain-specific model building.

ExSample [113] facilitates exploration through videos by prioritizing which frames to process with a pretrained model. ExSample assumes that the pretrained model used to answer queries is expensive, so it minimizes latency by adaptively sampling frames to apply the model to. While VOCALExplore also samples video segments, its task is to find video segments that should be labeled to train a high-quality domain-specific model. Once VOCALExplore trains a domain-specific model, ExSample could be used to find events of interest by selectively applying the model over unlabeled video segments.

Cloud vendor offerings. Amazon Rekognition [6], Google Cloud Video AI [9], and Azure Video Indexer [7] automatically index videos with common objects, scenes, or activities. They also enable training new models for custom labels, but in contrast with VOCALExplore they do not provide support to find examples to label. Rather, they expect users to come with labeled examples of the objects or activities they want to train on.

Data exploration. Current video browsing systems such as Cineast [127] and Vitrivr-Explore [71] are optimized for known-item search (e.g., based on an example clip or sketch) rather than exploration. Further, their query processing engines require the user to specify which features to use and how to weight them, whereas VOCALExplore automatically picks appropriate features for each dataset.

Lancet [168] combines active learning and semi-supervised learning (via label propagation), and it is designed for complex data that is represented by embeddings. While VOCALExplore uses pretrained models to extract embeddings for unlabeled data, Lancet jointly learns an embedding model and classifier. This requires tuning for each dataset to achieve good results and is expensive because embeddings must be updated when the model is retrained.

VQL [155] enables video exploration using the outputs of pretrained models, however it assumes the model is from the same domain as the target exploration, whereas VOCALExplore assumes no such model exists. Forager [8] enables efficient exploration and domain-specific model training over images or individual video frames rather than video clips. Further, it does not automatically pick a high-quality feature extractor for an arbitrary dataset. AIDE [47] is an active learning-based system for interactive data exploration over relational data rather than videos. It finds a SQL query over the data's attributes that returns the subset of data that is of interest to the user. While VOCALExplore trains a domain-specific model using the labels provided by the user, AIDE trains binary classifiers that mark data points as "interesting" or not.

Zero-shot ML and unsupervised learning. Image-language models capable of zero-shot inference over images and text have recently proliferated, such as CLIP [123]. However, as we show in Section 5.4, embeddings from video models outperform CLIP on datasets where the labels cannot be determined by looking at a single frame, such as deer activity classification. Thus far there has been limited work to develop video-language models. VideoCLIP [158] is capable of zero-shot inference over videos, however it performs poorly on the datasets we

evaluate VOCALExplore on (described in Table 5.2), achieving a macro F1 score of 0.04 for DEER and 0.33 for K20. Given the low zero-shot accuracy of current video-language models, it is necessary to implement domain-specific models.

VOCALExplore could be extended to incorporate unsupervised learning to leverage the entire dataset, however current techniques for videos [52] require training a large model and repeatedly extracting updated feature representations from videos, which is too slow for our goal of an interactive system.

AutoML. VOCALExplore shares similarities with AutoML systems [68] as it supports training models by automatically selecting a feature extractor and sampling data. VOCALExplore, however, does not attempt to maximize model quality via techniques that traditionally fall under the umbrella of AutoML such as feature engineering [150, 89, 162], data augmentation [169], hyperparameter optimization [80], or model selection [92]. Instead, it rapidly produces an initial model with minimal user-perceived latency. While VOCALExplore selects between possible feature extractors, this problem is different from that of traditional feature engineering. The input to VOCALExplore is pixel data and multiple candidate feature extractors that produce feature vectors, and the output is the feature extractor whose generated features lead to the best model for the given domain. In contrast, feature engineering takes as input features applicable to the domain and produces as output transformed features that improve model quality.

Chapter 4

TASM: A TILE-BASED STORAGE MANAGER FOR VIDEO ANALYTICS

In this chapter we address the research question of how to build efficient storage managers for video database management systems (VDBMSs), particularly when query workloads request spatial subregions of frames. We study this question through the design, implementation, and evaluation of TASM, which is a storage manager for VDBMSs that optimizes the physical layout of videos to enable spatial random access into frames. TASM adjusts the layout of videos based on the observed query workload to reduce query execution latency caused by decoding video frames to pixels. The work presented in this chapter appeared in ICDE'21 [41].

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 [103, 167], retail store planning [81], and drone analytics [149, 148]. This has led to a class of database systems specializing in video data management that facilitate query processing over videos [170, 82, 121, 73, 159, 81].

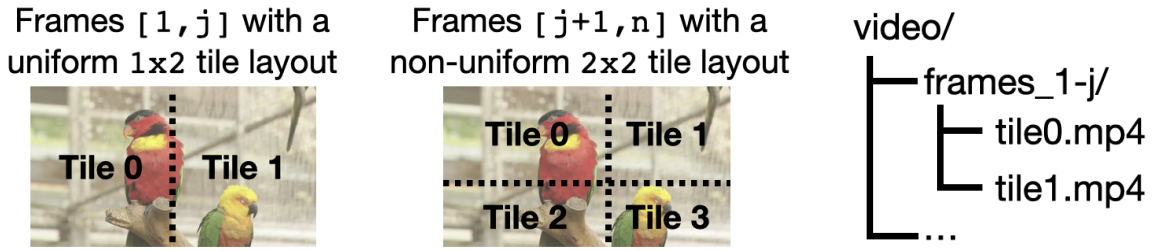
A query over a video comprises two steps. First, read the video file from disk and decode it. Second, process frames to identify and return pixels of interest or compute an aggregate. Most systems, so far, have focused on accelerating and optimizing the second step [73, 159, 81, 20], often assuming that the video is already decoded and stored in memory [81, 82, 104], 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 [104] or a specialized model [82] 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 [86].

In this chapter, 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 [103] or extracting image patches for vehicle type recognition [103] only need to operate on the parts of the frame containing vehicles. Privacy applications such as blurring license plates and faces [151] or performing region of interest-based encryption [14] 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 Figure 4.1, 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



(a) (b) (c)

Figure 4.1: 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 $[1, j]$.

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 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 chapter, 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 [77, 61] and online indexing [27]. 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 [40], 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 [86]. 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., [81, 20]).

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

- We develop TASM, 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.

4.1 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.

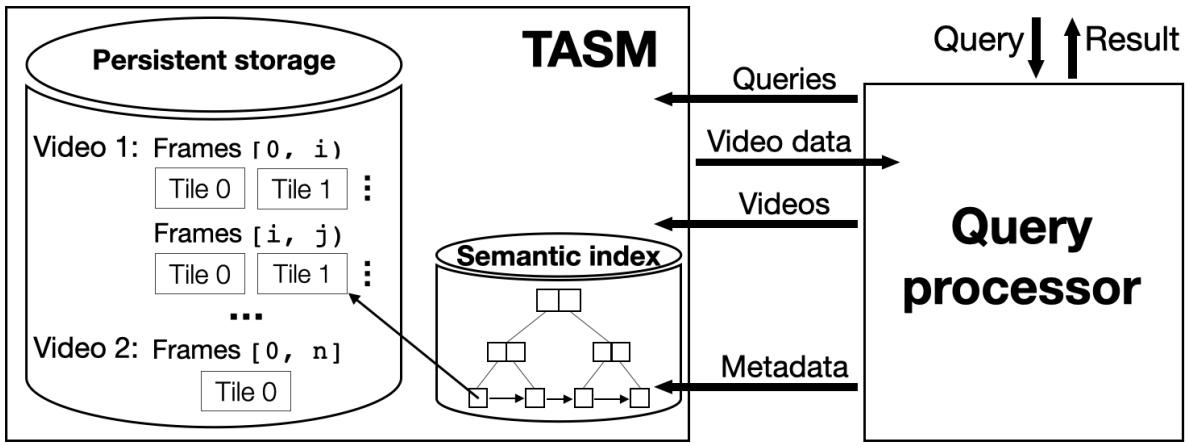


Figure 4.2: Overview of how TASM integrates with a VDBMS.

Table 4.1: TASM API.

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

Figure 4.2 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.

4.1.1 TASM API

TASM exposes the access methods shown in Table 4.1 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=($label$='*car*')$\vee$($label$='*bicycle*') 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.

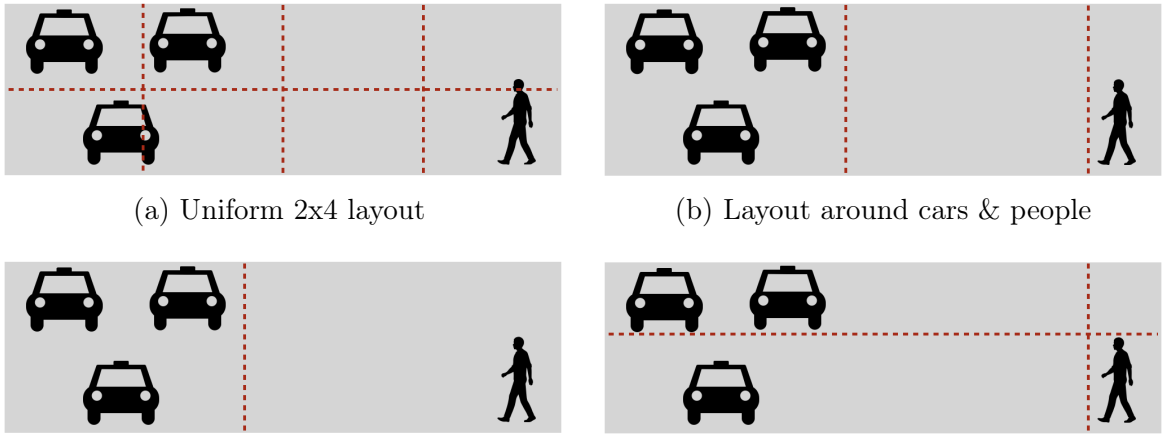
4.1.2 Semantic index

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 4.2, 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.

4.1.3 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. Figure 4.3 shows these different tiling strategies on an example frame.



(a) Uniform 2x4 layout

(b) Layout around cars & people

(c) Tile layout around cars

(d) Tile layout around people

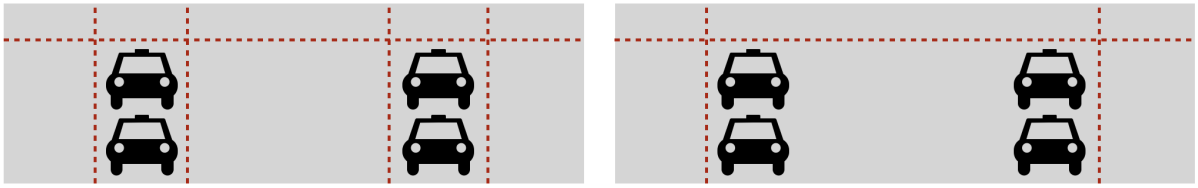
Figure 4.3: Various ways to tile a frame. (a) is a uniform layout, while (b)-(d) are non-uniform layouts. Depending on which objects are targeted, different layouts will be more efficient.

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 Figure 4.3a 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 Figure 4.6b.

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



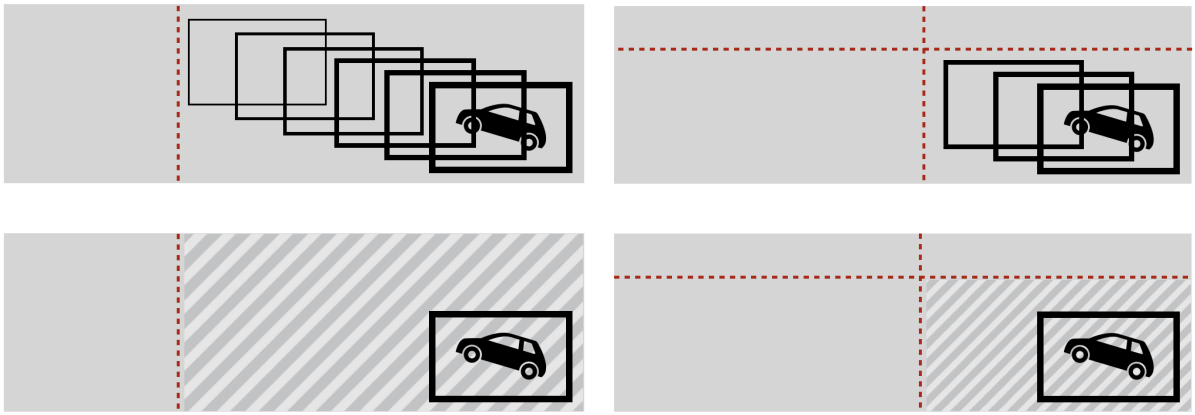
(a) Fine-grained tiles (b) Coarse-grained tiles

Figure 4.4: Non-uniform tile layout around cars using (a) fine-grained tiles, or (b) coarse-grained tiles.

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.

Figure 4.4 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 (Figure 4.4b), it places all B within a single, large tile. For fine-grained tiles (Figure 4.4a), 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 4.2.1.

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 Figure 4.3d, is more efficient than a layout around both cars and people (Figure 4.3b). We explain how TASM makes this choice in Section 4.2.



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

Figure 4.5: (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.

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 Figure 4.5. 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 Figure 4.9.

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.

Data storage and retrieval

TASM stores each tile as a separate video file, as shown in Figure 4.1. 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 2.1). This stitching process can also be used to efficiently convert the tiles into a codec-compliant video that other applications can interact with.

4.2 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 4.2.2. 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 4.2.3. Finally, TASM also creates an efficient, initial tiling before any queries are executed as we present in Section 4.2.4.

4.2.1 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 2.1), 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 2.1. If a SOT is not tiled, then $s_i{\mapsto}\omega$, where ω refers to a 1×1 tile layout. 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 4.1.3. For example, PARTITION$(s, \{car, person\})$ refers to creating a layout around cars and people, as in Figure 4.3b.

TASM implements a "what-if" interface [31] 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 4.3.2. 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 4.2.2. In contrast, tiling yields large improvements when objects are sparse. Figure 4.10 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 4.2.3 converges to such good layouts over time as queries are executed. Figure 4.8 also shows how object densities affect performance.

4.2.2 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 Figure 4.9 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 Figure 4.6a), and that fine-grained layouts outperform coarse-grained layouts (see Figure 4.8). 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=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 4.1.3, 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 experiments we find α=0.8 to be a good threshold. As shown in Figure 4.10, 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.

4.2.3 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 [27]. 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

Algorithm 1 Pseudocode for incrementally adjusting layouts

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

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 4.3.3 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 Figure 4.8, fine-grained tiles lead to better query performance than coarse-grained tiles when queries target new types of objects ($\text{PARTITION}(s, O'), O' \cap O_{q_j} = \emptyset$). 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 [40, 27]. 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.

Algorithm 1 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 4.2.1: $\Delta(q, L, L') = C(s, q, L) - C(s, q, L')$. 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 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 Figure 4.8b), so δ^0_{car} decreases. TASM retroactively accumulates regret for the new layouts. The accumulated regret for PARTITION$(s_0, \{person\})$ will be similar to δ^0_{car} because it would accelerate q_1 and hurt q_0. 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 [27], and we find it generally works well in this scenario, as shown in Figure 4.11. 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.

4.2.4 ROI tiling

Initially, nothing is known about a video. As we discussed in **??**, 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 [86]. 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 [104, 82], 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 [20].

Table 4.2: Datasets used to evaluate TASM

Video dataset	Duration (sec.)	Res.	Per-frame object coverage (%)	Frequently occurring objects
Visual Road [66]†	540–900	2K, 4K	0.06–10	car, person
Netflix public [97]	6	2K	0.32–49	person, car, bird
Netflix OS [131]*	720	2K, 4K	25–45	person, car, sheep
XIPH [4]	4–20	2K, 4K	2–59	car, person, boat
MOT16 [110]	15–30	2K	3–36	car, person
El Fuente [88]	480 (full) 15–45 (scenes)	4K	1–47	person, car, boat, bicycle

† Synthetic videos * Both real and synthetic videos

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 [2]. For example, specialized background subtractor modules can run at over 20 FPS on low-end hardware [165]. 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 4.3.5, an initial ROI layout in combination with semantic predicate pushdown can significantly accelerate the full scan phase of object detection queries while maintaining accuracy.

4.3 Evaluation

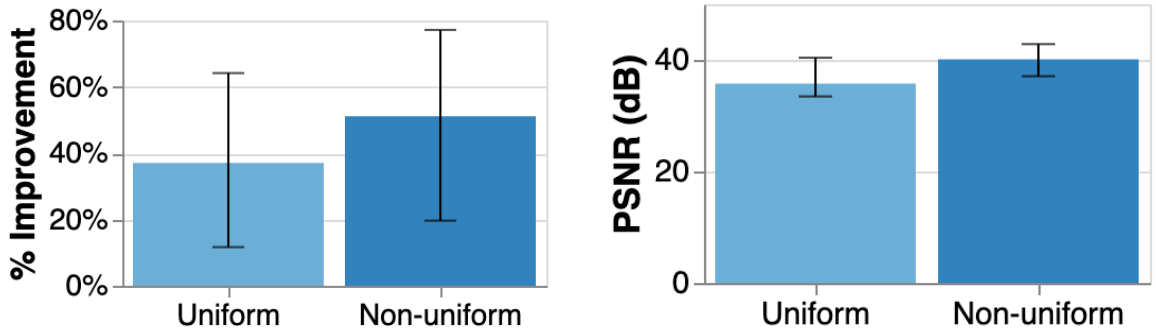
We implemented a prototype of TASM in C++ integrated with LightDB [65]. TASM encodes and decodes videos using NVENCODE/NVDECODE [117] with the HEVC codec. We perform experiments on a single node running Ubuntu 16.04 with an Intel i7-6800K processor and an Nvidia P5000 GPU. Our prototype does not parallelize encoding or decoding multiple tiles at once. We use FFmpeg [23] to measure video quality.

We evaluate TASM on both real and synthetic videos with a variety of resolutions and contents as shown in Table 4.2. 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 [66]) 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 [88] 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 [124], except for the MOT16 videos where we use the detections from the dataset [110]. We store the semantic index using SQLite [5], and TASM maps bounding boxes to tiles at query time.

The queries used in the microbenchmarks evaluated in Section 4.3.1 and 4.3.2 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 4.3.3 additionally include a temporal predicate (i.e., "`SELECT o FROM v WHERE start < t < end`"). [1] 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

[1] 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.



(a) (b)

Figure 4.6: (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.

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 4.3.2. 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.

4.3.1 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. Figure 4.6 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 4.3.2 and how it selects the optimal tile layout in Section 4.3.2 and Section 4.3.3.

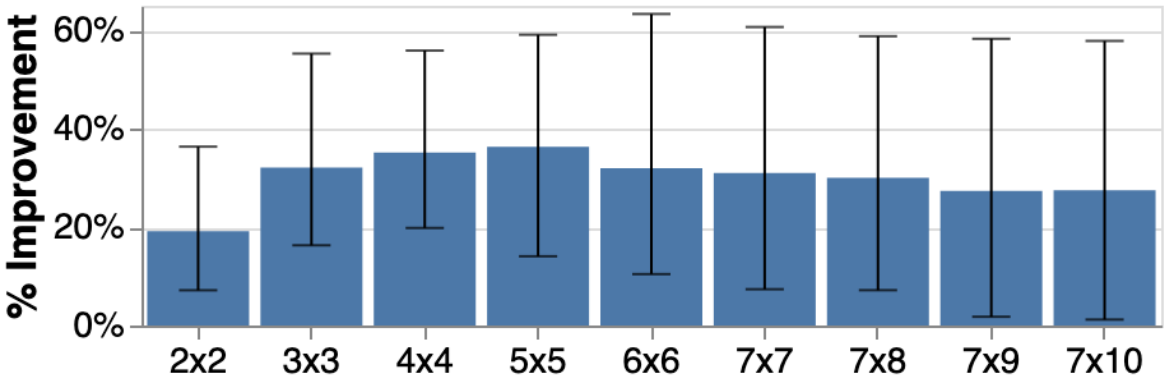


Figure 4.7: This figure shows improvement in query time achieved with various uniform layouts compared to the untiled video.

Figure 4.6a 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.

Figure 4.6b 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 [101], and videos with values $\geq$ 40 dB are perceived to have good quality [146, 97]. PSNR was computed over the entire tiled video stitched using homomorphic stitching [65] 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.

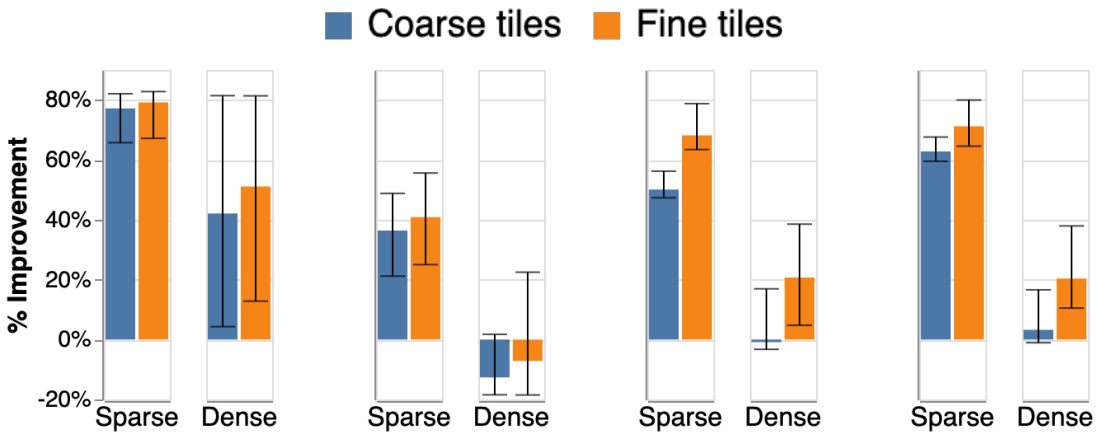
4.3.2 Microbenchmarks

Uniform tiles

We dig deeper into the results of Figure 4.6 and show in Figure 4.7 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. Figure 4.7 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.

Non-uniform tiles

The performance of non-uniform layouts depends on the objects queries target and the objects considered when designing the tile layout. Figure 4.8 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 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 Figure 4.3b). We further classify videos as *sparse*, where the average area occupied by all objects in a frame is $<20\%$, or *dense*, where it is $\geq 20\%$. Figure 4.8 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.



(a) Same (b) Different (c) All (d) Superset

Figure 4.8: 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 ≥20% in "dense" videos.

Figure 4.8 shows that tiling generally improves performance in sparse videos more than dense videos, and tile granularity has the largest impact when objects are dense. Figure 4.8a shows that when the tile layout is constructed around the query object, both coarse- and fine-grained tiles significantly improve query performance. Figure 4.8b 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 4.2.2, which we evaluate in Section 4.3.2. 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.

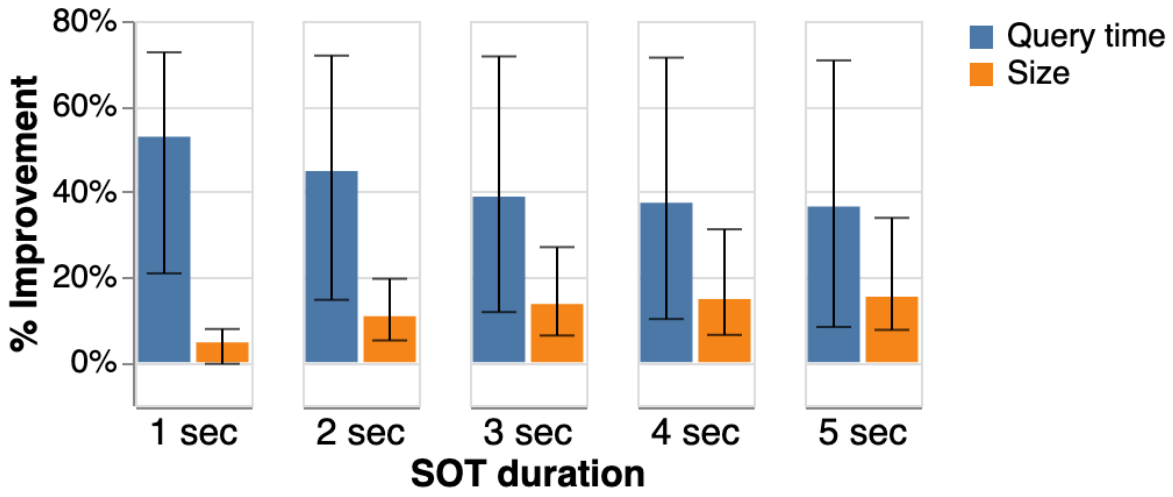


Figure 4.9: This plot shows the effect of SOT duration on query time and storage cost. Tiled videos were encoded with fine-grained tiles and a GOP length equal to the SOT duration.

Figure 4.8c 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. Figure 4.8d 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 2.1).

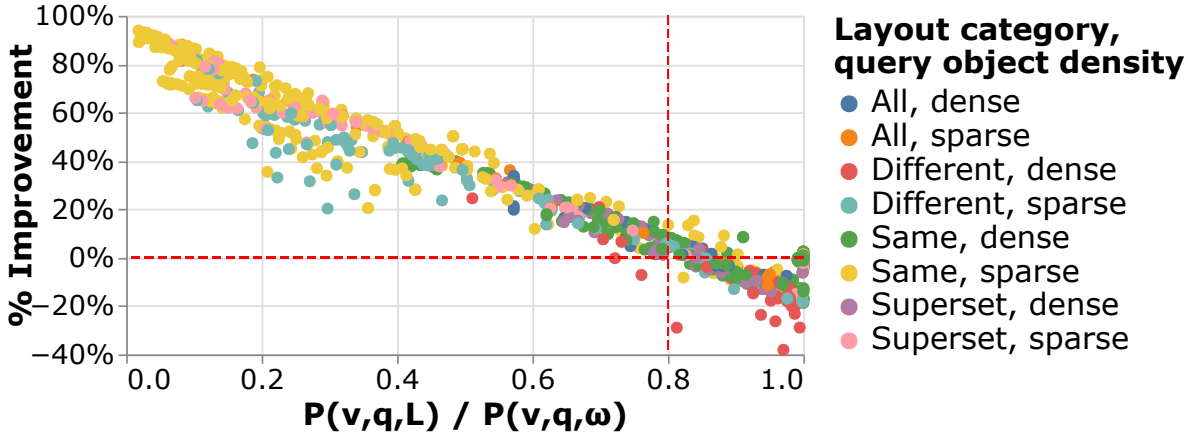


Figure 4.10: Ratio of the number of pixels decoded with a non-uniform layout to the number decoded without tiles vs. performance improvement. Each point represents a video, query object, and non-uniform layout. Points below the horizontal line at 0% represent cases where queries ran more slowly on the 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.

Figure 4.9 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.

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. Figure 4.10 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 4.2.1. 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).

4.3.3 Incremental tiling

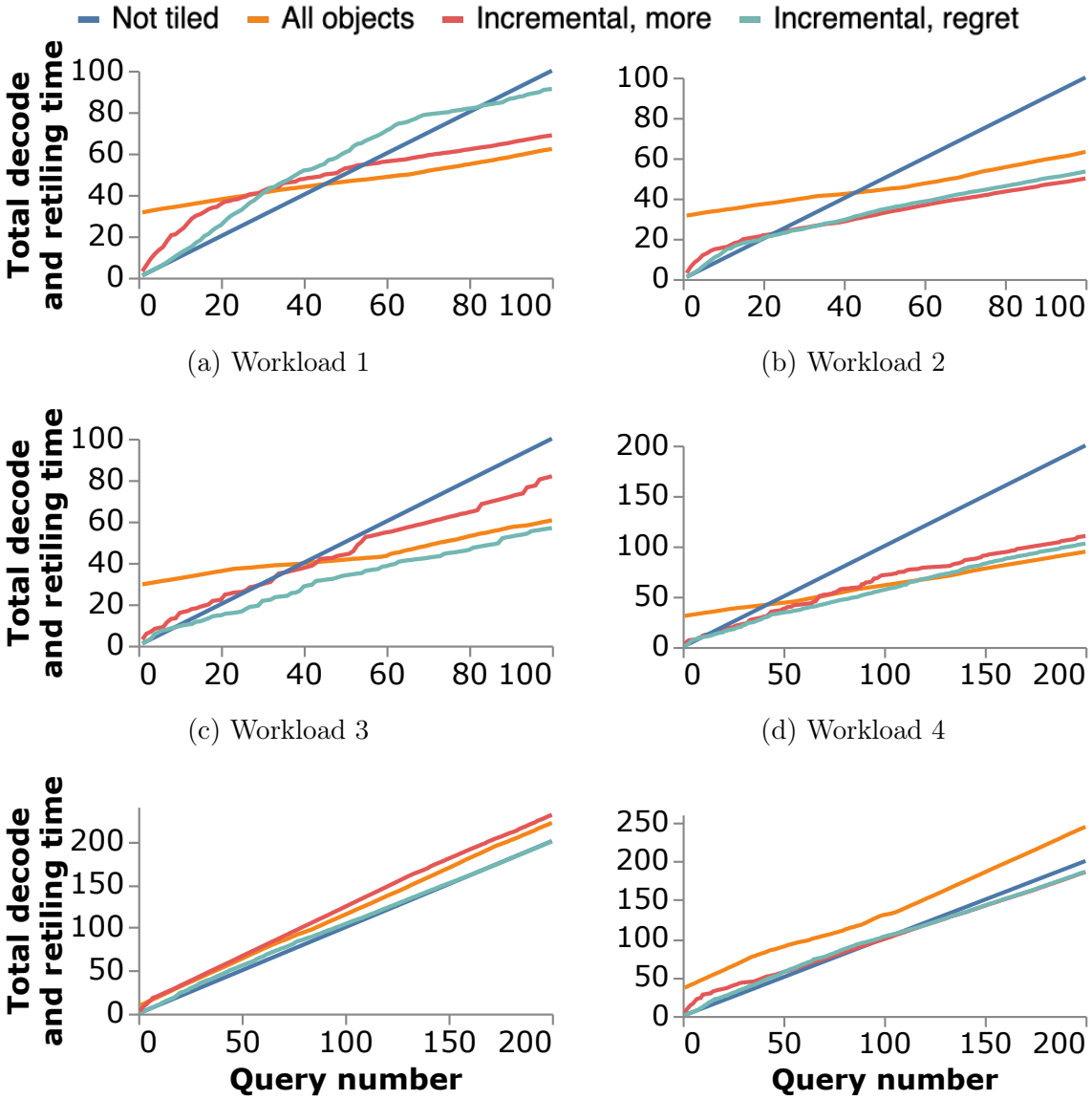
We next evaluate strategies for incremental tiling over various subframe selection workloads, which we construct to represent possible query patterns over videos. The baseline strategies are not tiling the video ("Not tiled") and tiling around all detected objects before queries are processed ("All objects"). 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 4.2.3 ("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 4.2.2). TASM estimates the regret using the cost function described in Section 4.2.1. 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 Figure 4.11 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 Figure 4.11 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 Figure 4.11a 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.



(a) Workload 1

(b) Workload 2

(c) Workload 3

(d) Workload 4

(e) Workload 5

(f) Workload 6

Figure 4.11: Cumulative decode and re-tiling time for various workloads. Values are normalized to the time to execute each query over the untiled videos.

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 Figure 4.11b, 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 Figure 4.11c, 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 Figure 4.11d, 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 Figure 4.11e, 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.

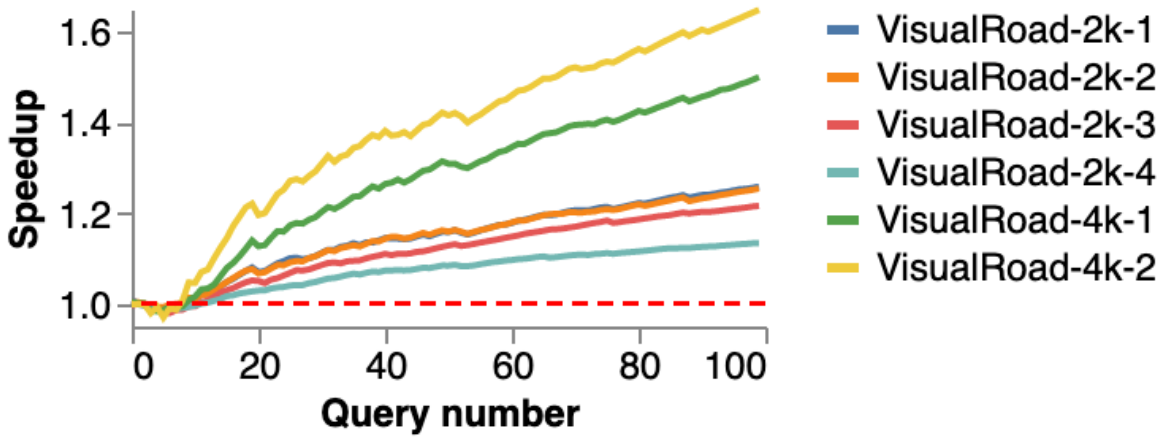


Figure 4.12: Speedup achieved with TASM over the Visual Road object detection workload. The lines show the median speedup over six orderings of the queries.

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 this criteria. The queries are uniformly distributed, and each targets the same object class over one second. As shown in Figure 4.11f, both incremental strategies eventually achieve layouts that perform better than not tiling. "All objects" performs poorly because objects in these videos are dense.

4.3.4 Macrobenchmark

Beyond the decoding benchmarks, we also evaluate TASM's performance on an end-to-end workload from the Visual Road benchmark [66], 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

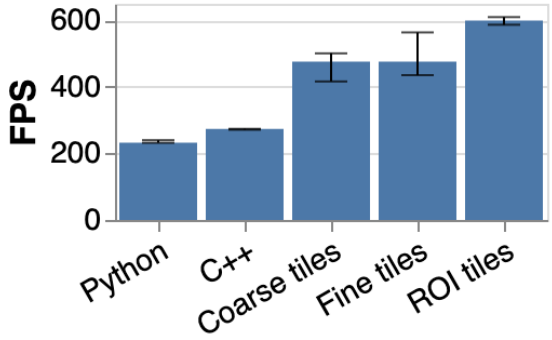


Figure 4.13: Specialized model preprocessing throughput

detect objects by running YOLOv3 [124] 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. Figure 4.12 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.

Table 4.3: Tiled 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

4.3.5 Object detection acceleration

We now evaluate TASM's ability to accelerate the full scan phase of object detection queries, as described in Chapter 4. One system that uses specialized models during the full scan phase is BlazeIt [81]. 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.

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 [25], and Numpy [63] ("Python" in Figure 4.13). We reimplemented this using C++, NVDECODE [117], and Intel IPP [79] 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 [25] to detect foreground ROIs on the first frame of each GOP. This is a throughput that recent mobile devices are known to operate above [165], and therefore it would be possible for this step to be offloaded to a compute-enabled camera as discussed in Section 4.2.4. 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").

Figure 4.13 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 4.3, models trained and evaluated on tiles show similar accuracy to full frame training within each day.

4.4 Summary

In this chapter we presented TASM, a tile-based storage manager for VDBMSs. TASM encodes videos using *tiles* to support efficient spatial random access into video frames. By enabling spatial random access, TASM reduces the decoding and preprocessing bottleneck for query workloads that operate over spatial subsets of frames, such as queries performing object detection only over the regions of the frame showing a crosswalk. TASM automatically optimizes and adjusts the tile layout for groups of frames over time as it observes the query workload and learns where in frames objects are located.

We evaluated TASM on videos from a variety of domains with varying resolutions and object density within frames. Our experiments showed that the tile layouts picked by TASM speed up subframe selection queries by an average of 51% and up to 94% while maintaining video quality. TASM also accelerates the full scan phase of object detection queries by up to 2× by tiling around cheaply-computed regions of interest.

Chapter 5

VOCALEXPLORE: PAY-AS-YOU-GO VIDEO DATA EXPLORATION AND MODEL BUILDING

Current VDBMSs and storage managers such as TASM (discussed in the previous chapter), presume access to an ML model that can identify objects and activities within video clips. These "oracle" models are integral to query processing over videos, as discussed in more detail in Section 3.2. However, many video datasets come from domains without such a pretrained model, and therefore these users cannot benefit from the capabilities of VDBMSs to query the contents of their videos.

This chapter thus investigates the research question of how to address the challenge of supporting early data exploration and domain-specific model building for video datasets that do *not* have an off-the-shelf oracle model, as described in Section 1.2. The work presented in this chapter will appear at VLDB'24 [43].

Increasingly many scientific domains rely on video data, which is information dense and relatively easy to collect. Examples include wildlife monitoring [45, 69], traffic analytics [15, 62], and many others [121, 56, 134, 140, 105]. Powerful libraries [25, 49, 106, 10] and data management systems [81, 16, 20, 113] exist to support users in storing, processing, and querying this video data. A key problem, however, is that those systems assume the user is already familiar with their data and, typically, already has one or more machine learning (ML) models to extract the desired information from the data. In speaking with scientists at the University of Washington, we find that this is frequently not the case. Instead, scientists collect data *en masse*, but then struggle to explore it, understand it, build ML models for it, and finally use it to answer their scientific questions.

Consider, for example, a scientist who wishes to understand the behavior and activity patterns of animals in the wild using cameras attached to animal collars (we describe this example in greater detail in Section 5.1.1). These cameras may easily produce terabytes of data. Because scientists mainly want to identify *activities* (as opposed to species, a well-understood problem [22]), there exists no off-the-shelf, pretrained model that can be used to process and extract meaningful data from this dataset. To develop a domain-specific model, scientists first need to familiarize themselves with their data and develop a vocabulary of activities within.

Existing tools do not adequately aid users in performing *early data exploration*—especially users who are not experts in ML—despite this being a critically-important component of end-to-end data management. Existing video browsing systems [72, 127] focus on *known-item search*, which presumes the user already knows what they are looking for and do not support building a domain-specific model. Lancet [168] proposes to support users in building domain-specific models over unstructured data by combining active learning with embedding training. However, their technique requires knowledge of ML tuning to achieve good performance on an arbitrary dataset, and it requires repeated, expensive processing over the entire dataset as the the embedding model is updated.

In this chapter, we present the design, implementation, and evaluation of VOCALExplore, a system that fills this gap and supports users with early video data exploration, labeling, and model building. It is a part of our larger VOCAL system [42]. In our example, the user only needs to point VOCALExplore at their data and they can *immediately* begin exploration. Immediate interactivity is a key goal of VOCALExplore. The user only invests more effort as they see results, which is important for new domains when the user may be uncertain about the labels they wish to use and whether a good model is even possible for their data and desired labels. A key contribution of VOCALExplore is to support such initial exploration with a "pay-as-you-go" design, which avoids expensive preprocessing phases. Instead, VOCALExplore processes data incrementally as the user explores it and provides increasingly accurate results as users put more effort towards exploration and labeling. VOCALExplore enables an iterative

workflow: At each step, the user either specifies which video segments they want to view or lets the system select video segments. As they watch videos, they can choose to annotate them with new or existing labels. When VOCALExplore chooses video segments for the user to view and label, it samples them in a way that yields good model quality, while avoiding extra costs when not necessary. VOCALExplore also decides which feature representation to use for the given data. Finally, VOCALExplore does the above while hiding all significant sources of user-visible latency, providing fast response times to data exploration requests.

There are several challenges in designing a system like VOCALExplore. First, VOCALExplore brings together techniques from across the ML community that are required to support end-to-end video data exploration and model building—from video sampling to feature extraction to building models on video data. VOCALExplore combines these into a system wrapped behind a data exploration interface, which does not require any ML knowledge or tuning from the user, nor any expensive preprocessing steps.

Second, we design VOCALExplore as a pay-as-you-go system: i.e., it receives user input incrementally and it must produce results incrementally as well, all without long preprocessing phases to maintain the low-latency data exploration promise. At each iteration, VOCALExplore must decide what set of video segments the user should label next and how to train the best model on top of these labels. The user specifies a labeling budget equivalent to how many videos the user is willing to label. While the ML community has proposed many active learning acquisition functions [135], there is evidence that no one technique is the best, and they often perform no better than random sampling as shown in [87] as well as in our evaluation (Figure 5.3). Further, active learning-based acquisition functions are more expensive than random sampling because they require preprocessing the dataset. To address this challenge, our system dynamically selects either random sampling or an active learning-based acquisition function based on observed class imbalance in the dataset. VOCALExplore always starts with random sampling because it is expected to perform well over uniform datasets, and it requires no preprocessing. It then switches to a more expensive active learning-based acquisition function if it observes sufficient skew in the

labels. When it switches to active learning, VOCALExplore incrementally processes videos to build a candidate set over which the active learning algorithm can execute, again avoiding an expensive preprocessing step.

While it is common today to train video models using pretrained models as feature extractors, there is a lack of research exploring how to choose the best one for a new dataset. Therefore, before we begin to train a domain-specific model using the user provided labels, we are faced with the technical challenge of deciding which pretrained feature extractor to use. We show that the accuracy of domain-specific models depends on the chosen feature extractor. To address this challenge, VOCALExplore starts with a set of candidate pretrained models to be used as feature extractors. It frames feature selection as a rising bandit [96] problem to dynamically converge on the best features for a given dataset during early labeling iterations—again avoiding a separate feature selection phase—and instead integrating feature selection into the data exploration process. Note that we use the term "feature selection" to denote picking a feature extractor, rather than selecting a subset of a feature vector.

The third challenge is supporting the functionality described above with low user-visible latency to make data exploration interactive. VOCALExplore relies on many tasks that have non-trivial latency (e.g., training a model and extracting feature embeddings from encoded videos), and naive strategies to minimize latency risk hurting model performance (e.g., eliminating model training latency by making predictions using a model trained many iterations ago). VOCALExplore addresses this challenge by using idle time to perform tasks while the user is occupied labeling videos. While the idea of leveraging background processing is not new, the key contribution of this chapter lies in identifying *which* tasks to execute in the background and *when* to launch them in order to achieve a model quality that is as similar as possible to a serial execution of all tasks, all while maximally reducing user-visible latency.

In summary, VOCALExplore makes the following contributions:

- We design a video data exploration and labeling system that brings together state-of-the-art ML methods and wraps them with a simple data exploration interface that does not require any ML knowledge from users (Section 5.1).

- We develop an Active Learning Manager (ALM) that produces high-quality models by dynamically selecting the appropriate acquisition function and best feature extractor for each dataset (Section 5.2).

- We develop a Task Scheduler to ensure VOCALExplore produces high-quality models without significant user-visible latency (Section 5.3).

We evaluate VOCALExplore on standard and domain-specific datasets (Table 5.2). Our experiments show that VOCALExplore can achieve model performance that matches the best combination of acquisition function and feature with no preprocessing, and a user-visible latency of less than one second per labeling iteration. VOCALExplore does this while automatically deciding what features to use and how to sample video segments to be labeled.

5.1 System Overview

In this section, we present the API of VOCALExplore, user workflow, and overall system architecture.

5.1.1 Motivating example

We first motivate VOCALExplore by describing the use case of the ecologists we partnered with who study the behavior of deer in the wild [45]. The scientists seek to understand how much time the deer spend on different activities (e.g., eating or traveling). To study these questions, the ecologists attached GoPro-style cameras to collars on the deer. These cameras collected video data for two weeks before the collars automatically fell off the deer.

Once the ecologists collected the cameras, they had access to a large quantity of video data (1.4 TB across $800k$ video files) that, were it labeled, would enable them to analyze their research questions. An ideal solution for these scientists would be to automatically label the videos using a machine learning model. However, no pretrained model exists for this domain-specific task. Therefore, the ecologists manually labeled a sample of the videos by temporally sampling video clips from the morning, midday, and evening, and performed their analysis on top of these labeled samples [45].

This manual labeling process is tedious, and analyzing the labeled samples is limiting, especially when the fraction of labeled data is small. As an alternative, the scientists could have manually trained a domain-specific model. This is, however, challenging for the reasons already enumerated, and because the scientists are not experts in ML. Next, we describe how VOCALExplore supports scientists to easily train a model over their videos.

5.1.2 API and user workflow

Workflow. Here we describe the high-level workflow users follow when using VOCALExplore. Users load their video data by specifying a set of video paths. Users can immediately start exploring and labeling their data because VOCALExplore performs no preprocessing. During this exploration, VOCALExplore samples video segments for the user to label. Initially, it randomly returns videos for the user to explore. Once the user has provided some initial labels (in the prototype, $\geq$5 labels), VOCALExplore additionally returns the predicted labels for each produced video segment. At any time, the user can view any subset of the video data together with VOCALExplore's predictions for those videos. The user can provide corrected labels for any errors they notice.

API. The API of VOCALExplore is shown in Table 5.1. **Watch** enables a user to view a video stream within a specified time window. VOCALExplore returns a sequence of consecutive video segments labeled with the activities that the system detects. Initially, labels are null. **Explore** enables system-directed exploration to efficiently build a high-quality domain-specific model. VOCALExplore returns videos (along with their predictions) that

Table 5.1: VOCALExplore API.

Method	Parameters	Description
WATCH	(vid, start, end)	Returns a stream of video segments from *vid* between *start* and *end* with predicted labels
EXPLORE	(B, t, label=None)	Returns B video segments each of duration t with predicted labels
ADDLABEL	(vid, start, end, label)	Saves the label as metadata
ADDVIDEO	(path)	Saves the video as a candidate for labels and predictions and returns its vid

when labeled will most improve model performance. EXPLORE optionally takes a specific label that causes VOCALExplore to return videos that will most improve its predictions for the specified class. When a user views video segments they can add labels using the **ADDLABEL** method.

5.1.3 System architecture

To enable the above workflow, VOCALExplore must support the following functionalities: sample selection (i.e., produce video segments needing labels when the user calls EXPLORE); model training and inference (i.e., train a model using the labels provided by the user up until that point and produce labels for unlabeled videos); and feature extraction (i.e., what inputs are used for model training and inference). Figure 5.1 shows the overall architecture of VOCALExplore that supports these functionalities. The Active Learning Manager performs sample selection, the Model Manager performs model training and inference, and the Feature Manager performs feature extraction. Additionally, VOCALExplore includes a Storage Manager to manage metadata and intermediate results, and a Task Scheduler to coordinate the activities of these components.

The Active Learning Manager (ALM; Section 5.2) and the Task Scheduler (Section 5.3) are the novel components and the core contribution of this paper. We defer a detailed discussion on their associated challenges to the following sections; this section focuses on the overall

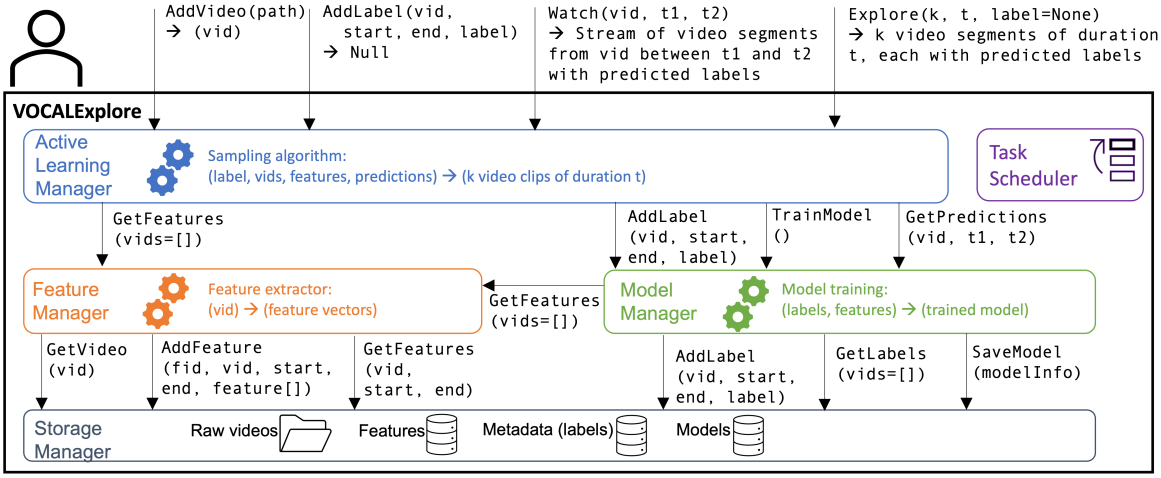


Figure 5.1: VOCALExplore architecture. The Task Scheduler coordinates the activities of the various managers.

architecture and outlines how the various components interact together. Figure 5.1 illustrates the primary methods implemented by VOCALExplore. Many system calls optionally operate over a set of videos, represented by `vids`. For example, the Model Manager trains models over features and labels from multiple videos.

Storage Manager (SM). The SM stores and retrieves all persisted data, which includes video metadata (e.g., path, duration, start time), labels, features, and models. The SM uses off-the-shelf components.

Feature Manager (FM). The FM returns feature representations of video segments. These feature vectors are used by the ALM to decide which video segments the user should label as well as by the Model Manager to perform training and inference. Features are represented by d-dimensional vectors in $\mathbb{R}^d$, and each vector is associated with some time period $(start, end)$ within a video.

Model Manager (MM). The MM trains models using the user-specified labels and performs inference on these models to return predictions. Given a video (`vid`), and time-interval $[t_1, t_2]$, the MM outputs a probability distribution across possible labels for that video

segment. Our prototype MM maintains one model per feature extractor. The MM trains a new model whenever requested to do so by the ALM and is non-blocking: while a new model is training, the MM serves requests for labels using the previously trained model.

Active Learning Manager (ALM). For each call to EXPLORE, the ALM picks B video segments, each of duration t, that the user should label next. For each call to WATCH and EXPLORE, the ALM invokes the MM to provide predictions for the video segments being returned. We further describe the ALM in Section 5.2.

Task Scheduler (TS). The TS coordinates the activities of the various managers to ensure low-latency responses to user-initiated API calls while maintaining high prediction quality. We describe the associated challenges and how the TS addresses them in Section 5.3.

5.2 Active Learning Manager

The Active Learning Manager (ALM) is a central component of our system responsible for selecting the video segments that the user should label. Recall that our system focuses on tasks where a user wishes to label a small number of video segments to build a model that can serve to label the rest of the video. The ALM must address several challenges. Most importantly, our system's goal is to provide pay-as-you-go results: i.e., for each new batch of user labels, the ALM strives to maximize model quality given the labels collected so far. The ALM cannot rely on a long preprocessing phase to accumulate a large number of labels or optimally select features for a new domain. Instead, the ALM generalizes the problem of active learning to not just choosing which video segments to label (and what method to use to perform that selection), but also *simultaneously* choosing which features to use for a new domain.

The first subproblem of selecting segments to label is an active learning problem. There are many proposed acquisition functions in the active learning literature (e.g., [135]). Our goal is not to design a new active learning algorithm, but to determine when the extra cost is worthwhile in a data exploration system. Because random sampling can achieve the best model quality in some settings [87] and is less expensive, the first challenge the ALM addresses

is distinguishing between when random sampling is sufficient and when an active learning acquisition function should be used for a given dataset. The key idea behind our approach is for the ALM to start with RANDOM, observe the label distribution, and dynamically switch to other acquisition functions if the evidence suggests that active learning will outperform RANDOM. Section 5.2.1 describes how the ALM chooses between these functions.

The second subproblem is feature selection. Video models use pretrained features as a starting point for new tasks. However, choosing the appropriate pretrained model from which to extract features is an open research question. We propose to dynamically select features to use for a given dataset. The key idea of our approach is to use a rising bandit method to comparatively evaluate feature quality during active learning as we describe further in Section 5.2.2. In contrast with feature engineering approaches [150, 89, 162], our problem is to produce a useful feature representation for the unstructured video data in the user's new domain rather than manipulating features to improve model performance.

Finally, the ALM solves both subproblems simultaneously. At each step, it makes the best decision for each independently. However, the samples selected by the acquisition function affect model performance (and therefore feature selection), and features affect the performance of active learning sampling. The ALM handles this interference by using decision methods that are tolerant to noise.

5.2.1 Acquisition function selection

We first discuss how the ALM solves the problem of acquisition function selection, where the acquisition function determines which video segments are selected to be labeled at any given iteration.

Problem. The ALM is given a set of video segments, $v \in V$. The video segments depict various activities $a \in A$, and these activities may be skewed, meaning that some appear more frequently than others. It is possible for a single video segment to contain multiple activities, or no activities. We are also given a labeling budget, B, which designates the number of video segments a user is willing to label. This budget is incremental and is not fixed. For example,

a user may initially set B=20 and then give VOCALExplore another B=10 if they are willing to label more. The ALM uses the labels to train a model M that predicts the activities appearing in each video segment. The prototype trains linear models using cross-entropy loss, which is often used for classification problems [107].

The ALM balances the two goals of maximizing model quality, (G1), and producing pay-as-you-go results, (G2). For G1, we consider the average model quality across all classes the user has applied to video segments. Because VOCALExplore also allows users to specify classes of interest, the ALM strives to improve model performance on these specific classes, when requested. The prototype maximizes the macro F1 score of the model, though other metrics could be used. For G2, the ALM strives for interactivity and low latency in response to API calls by avoiding expensive preprocessing steps that block user interactions.

Baselines

We consider the use of individual acquisition functions as baselines. The relative performance of any acquisition function depends on the dataset, but the cost of each function is partially determined by the inputs the function requires (the other component of cost is the processing done on top of the inputs).

The most naive strategy is RANDOM, which randomly selects the B video segments. This is cheap because its inputs are video metadata (e.g., duration) rather than features extracted from the video frames. However, if the activities in the dataset are highly skewed, then random sampling will not find many examples from activities that rarely occur, which hurts G1 because the model will perform poorly on these rare classes. Additionally, as we observe experimentally, random sampling over skewed data causes the user to label large amounts of the same activity type and very few rare activities. We posit that having the user label more diverse activity types is more in line with supporting users in early data exploration.

More sophisticated baselines use active learning techniques that take as inputs features, and possibly model outputs. These strategies require an expensive, one-time preprocessing step to extract features from all of the video segments V. Uncertainty-based techniques

additionally require performing inference over all $v \in V$. This preprocessing hurts G2 because it results in a large amount of initial latency, even if the user only makes a small number of API calls, and the feature extraction and inference tasks over all of the videos result in high latency for API calls. However, active learning acquisition functions can improve model performance over naive random sampling [133], especially for skewed datasets where random sampling will have low label diversity.

*Our approach (*VE-SAMPLE*)*

The ALM resolves these tradeoffs by casting acquisition function selection as a binary decision between RANDOM or an active learning-based acquisition function. It dynamically switches to a more expensive active learning function only when it is expected to improve model performance and label diversity. The ALM strategy, which we call VE-SAMPLE, initially uses random sampling to select the B video segments to be labeled because it is fast and requires no preprocessing (G2), and it performs well for uniform datasets (G1). VE-SAMPLE dynamically switches to active learning if it observes skew in the labels it collects. This results in better label diversity for the user and, more importantly, improves model performance on rare classes (G1).

Recall that EXPLORE may be called with or without a user-supplied label (see Table 5.1). For EXPLORE calls *without* a specific label, when our prototype switches to active learning it uses cluster-margin sampling [36] which combines uncertainty and diversity sampling. Briefly, the algorithm works as follows: it performs a one-time clustering of all examples to be reused at each iteration. Given a desired batch size of B examples, cluster-margin sampling first uses uncertainty sampling to select the kB examples where the model is least confident, where k is a hyperparameter. The algorithm then selects B diverse examples from these most uncertain ones by iterating round-robin over their clusters; it selects one example randomly from each cluster until it has a result set of size B. Our prototype also implements the greedy coresets algorithm [133] (CORESET), which is a density-based acquisition function that has

been shown to work well in a batch-labeling setting and is designed to find diverse examples. By default the ALM uses CLUSTER-MARGIN sampling for active learning because in our experiments it always performs at least as well as CORESET.

For EXPLORE calls *with* a specific label, VE-SAMPLE uses uncertainty sampling [95]. We follow the uncertainty sampling procedure described in [114] because the authors showed it performs well on rare classes. VE-SAMPLE uses a domain-specific model trained on all labels provided so far and finds video segments corresponding to features where the model is either highly confident or highly uncertain that a given feature vector should be assigned the specified label. Let n_a be the number of video segments labeled with activity a, and n_o be the number of video segments labeled with any activity other than a. Following [114], we pick the video segments with the most confident predictions when the number of positive labels for activity a is less than the number of negative labels ($n_a < n_o$). Otherwise, the ALM picks the video segments with the most uncertain predictions when $n_a \geq n_o$. The intuition behind this approach is to first train a model capable of identifying easy positives, and then to refine the model's performance near the decision boundary by picking hard positives and hard negatives.

To decide whether the labels are sufficiently skewed to switch to active learning, VE-SAMPLE uses the k-sample Anderson-Darling test [130] which is a statistical test for comparing discrete distributions. VE-SAMPLE compares the label distribution observed so far to a baseline uniform distribution and switches to active learning when $p \leq 0.001$. We use this small p-value because the label distribution is initially noisy when there are a small number of labels. We want to switch away from random sampling only when we are highly confident that the distribution is in fact skewed.

Other statistical tests are possible. For example, we could also say that a dataset is skewed if the imbalance ratio [119] (i.e., the ratio between the frequency of the majority and minority classes) is large. If there are k classes, and the multinomial distribution has parameters $p \in \Delta_k = \{p \in \mathbb{R}_+^k : \sum_{i=1}^k p_i = 1\}$, we can say a distribution p is skewed if $\min_i p_i < \frac{1}{mk}$ for some multiplicative threshold m. m is a lower bound on the imbalance ratio because the

majority class must have frequency $\geq 1/k$. For this frequency-based approach we set the p-value to be equal to an upper bound on the probability of incorrectly classifying a dataset as skewed. Details of how this bound is derived are described in Appendix A. The benefit of using the frequency test is that its p-value will not grow smaller solely based on an increasing number of data points if the dataset is not perfectly balanced. Whereas the Anderson-Darling test will return a small p-value for slight class imbalances (e.g., 51% class A and 49% class B) given sufficient labels, the frequency test with high probability will not detect this as skewed even in the limit of infinite labels. We show in Section 5.4.2 that this frequency-based test matches the F1 scores achieved when we use the Anderson-Darling test.

Interestingly, we empirically find that the VE-SAMPLE approach has the additional effect of producing a more diverse set of video segments for the user to label, compared with using random sampling alone. A diverse labeled set benefits model performance, but it also makes the labeling task more interesting for the user. Given $n_a, a \in A$, the number of labels for each activity type, we measure label diversity as $S_{max}=\frac{\max_{a \in A} n_a}{\sum_{a \in A} n_a}$, which represents the fraction of labels that come from the most-seen activity. A lower S_{max} indicates a higher diversity of labels. Other measures are possible.

Finally, the ALM addresses G2 by incrementally processing videos. The ALM extracts features from labeled videos to train models, and from sampled videos to make predictions, so the amount of processing is proportional to the amount of user interaction. For RANDOM, this requires only processing the videos that contain the B video segments returned from EXPLORE. However, when VE-SAMPLE switches to active learning, the active learning algorithm requires a set of candidate features. The ALM balances active learning quality and visible latency through a hyperparameter, X. When VE-SAMPLE is using active learning and the user requests B video segments from EXPLORE, VE-SAMPLE ensures this set contains features from X additional videos. We evaluate the impact of the choice of X on both latency and model quality in Section 5.4. As described in Section 5.3, the Task Scheduler hides the latency of this incremental processing so it does not affect interactivity.

5.2.2 Feature extractor selection

As discussed previously, VOCALExplore uses pretrained image and video models as feature extractors because they have a favorable cost/quality tradeoff (model inference is highly optimized on GPUs), and training a linear model on pretrained features is an accepted technique for training domain-specific models [51]. The MM trains one model per candidate feature; the model predicts which activities appear in each video segment.

Problem

We observe that the performance of feature extractors varies depending on the dataset and task. As we shall see in Figure 5.4, some feature extractors perform much better than others on a given dataset, and the best feature varies across datasets.

The ALM is responsible for finding a feature extractor that leads to high-quality models when trained over the user-provided labels. By default, VOCALExplore uses a pool of video and image pretrained models as candidate feature extractors. The prototype is designed to support video classification tasks and therefore uses pretrained classification models. However, a different set of candidate models would likely be needed for other labeling tasks, such as segmentation. To extract features for a particular video, the FM performs inference on sampled clips or frames (for video or image models, respectively) to extract feature vectors. Each feature vector is associated with a feature ID, video ID, and some time span corresponding to the input frame(s): $(fid, vid, start, end, vector)$.

The ALM must pick a feature to use at each step when it returns predictions for video clips because the feature determines which model is used to make predictions. The ALM must also pick a feature to use if VE-SAMPLE uses active learning (see Section 5.2.1). Picking a feature that performs poorly leads to incorrect predictions, and, in the case of active learning, suboptimal clip sampling. Therefore, the ALM dynamically selects the features to use based on the empirical performance on each dataset.

Naive strategies

A first naive strategy is to concatenate all of the possible features into a single, long feature vector. This has the benefit of not having to explicitly pick one feature, and, given enough labels, models will identify the vector elements with the most signal. However, this requires a large amount of compute resources to extract all features from all videos (as shown by the latency of the VE-LAZY (PP) strategy in Figure 5.7). The amount of compute needed to extract features grows linearly with the number of extractors and the number of videos. If there are V videos and F candidate extractors, the amount of compute is $O(V \cdot F)$. Further, we do not observe an improvement in performance over the best single feature, as shown in Figure 5.4.

A second naive strategy would be at each step to use the feature that is performing best. At each step, the ALM could extract *all* possible features from all labeled video clips, and then train a different model for each feature. It would pick the feature that has the best model performance, measured using k-fold validation or over a held out validation set. This second naive strategy has the same problem as the previous one of requiring all possible features to be extracted from videos. It also is inefficient to train and evaluate models for all possible features at every step.

Bandit strategies

While the ALM initially must explore all possible feature extractors as in the second naive strategy, we want to quickly converge on one of the best ones. Once a feature is picked, all compute resources can be dedicated to extracting just that feature from the remaining video segments, and models are only trained using that feature. The problem then becomes how to converge on one of the best features. On the surface, this appears to be a problem that can be solved by Multi-Armed Bandit (MAB) approaches: each feature extractor is an arm, model performance is the reward, and we want to exploit the feature extractors that lead to the best model performance. However, MAB techniques assume stationary reward distributions

(i.e., the reward for pulling an arm is independent of the number of times that arm is pulled). This is not true for our use case because model performance is expected to improve as the amount of training data increases. If a feature performs poorly in early rounds, we do not want to eliminate it solely based on early performance values because it is possible that it will improve once there are more labels.

Our setting is that of *Rising Bandits* [96]. Rising Bandits do not assume that the rewards for each arm are stationary; rather, they are assumed to be increasing in a concave manner as the arm is pulled. Under these assumptions, the expected performance of each arm after some number of examples can be bounded, and arms can be eliminated when the upper bound on their expected reward is lower than the lower bound for some other arm.

The original Rising Bandit algorithm [96] that the ALM adapts works as follows: The algorithm proceeds in a series of rounds. At each step, it computes the current model quality for each candidate feature. Then, it computes lower (l_f) and upper (u_f) bounds for the expected performance after T timesteps. The lower bound is taken to be the current value because we assume the quality increases over time. The upper bound ($u_f = l_f + \omega_f \times (T - t)$) is taken to be the lower bound plus some delta computed as slope (ω_f) multiplied by the number of remaining timesteps ($T - t$), which is a linearization at the current time t step evaluated at T. Because of the concavity assumption, the linearization is an upper bound on the true reward. Finally, features are eliminated when their upper bounds are below the lower bound of any other feature. Note that the algorithm from [96] was proposed in a different setting from ours and thus the guarantees do not directly transfer. In particular, the "reward" in our setting is the performance of the chosen arm with T points, while the "reward" in [96] is the performance of the chosen arm with however many points were allocated to that arm.

VOCALExplore adaptations to Rising Bandits

The ALM must resolve three challenges before applying the Rising Bandit framework. First, measured model performance is noisy. While it is expected to increase on average over time, individual time steps may have a decrease in performance if the added labels temporarily

make it more challenging for the model to distinguish classes. Second, measured model performance is not guaranteed to increase in a concave manner because the training set grows over time and because the ALM may switch to active learning from random sampling. Finally, the user does not initially have a labeled validation set, but the ALM still must reliably estimate model performance.

To resolve the first challenge of noisy performance data, the ALM performs smoothing on top of the measured values. The goal is to capture the trends in performance but avoid any temporary spikes or dips. The prototype uses exponential weighted moving average (EWMA) smoothing, but other techniques are possible. The prototype also waits 10 iterations before beginning feature selection because model performance is particularly noisy in early iterations when there are a small number of labels.

To resolve the second challenge of non-concave performance increases, the ALM uses the proposed solution from the Rising Bandits algorithm [96]. Recall that the algorithm computes the upper bound using the slope to estimate the value after some number of steps into the future. Rather than computing the upper bound using a slope over the current and immediately previous timesteps t and $t-1$, the ALM computes a smooth growth rate over a larger window of size C: t and $t-C$.

To resolve the final challenge of the lack of a validation set, the ALM estimates the performance of features using cross-validation. The ALM creates three train/test splits over the labels it has collected so far and averages the performance across these splits. While training and evaluating multiple models is more expensive than evaluating a single model over a held out validation set, the ALM only does this at the start of exploration when there are a small number of labels until it picks the best feature (which usually requires fewer than 150 labels in our experiments). Training linear models with a small number of examples is fast, so the additional overhead is limited. The prototype only evaluates k-fold validation over classes with at least three labeled instances to ensure each class is present in each training and test split.

While the original algorithm in [96] evaluates one arm at each time step, our modified algorithm evaluates all candidate features at each time step because the new labels provided by the user can be used to update the model for all features.

Hyperparameter setting

The hyperparameters the ALM uses for feature selection are: C (slope smoothing window), T (timestep used to compute the upper bound), and w (smoothing span for EWMA; α=2/(w+1)). As discussed in Section 5.4.3, the sensitivity of C and w is low; a range of values provide similarly good performance. This agrees with the findings of [96] that the performance of their algorithm is not sensitive to C. Therefore, the ALM uses a "moderate" amount of smoothing and sets w=5 and C=5.

T is the time point at which the upper bound is computed. Larger T values lead to higher upper bound estimates, therefore features are eliminated more slowly. Using a larger T value is more robust against non-concave performance curves because when the slope is small at early steps, the upper bound will still be high enough to not eliminate the feature before its slope later increases. However, larger T values require more compute power because a larger number of features will be extracted and evaluated for more steps. Therefore, our approach is to set T to a small value (e.g., $T \leq 50$) in resource-constrained settings. This may not lead to selection of the optimal feature, but our evaluation shows that one of the best features is still selected with high probability. In settings where resources are not constrained, T can be set to a larger value (e.g., T=100) because there are sufficient resources to evaluate more features for additional steps, and therefore allow the ALM more time to attempt find the single best feature (though, using a larger T doesn't *guarantee* finding the best feature).

5.3 Task Scheduler

The Task Scheduler is a priority scheduler that runs in the background and schedules VOCALExplore's tasks on the available compute resources. We consider a setting where there are limited resources, so only a subset of submitted tasks can execute at once. From

Section 5.2.1, goal G2 states that VOCALExplore should ensure interactivity and low latency in response to API calls. VOCALExplore is intended to support data exploration, so it needs to minimize any user-perceived latency because increased latency is known to decrease user interaction [98]. Naive and lazy scheduling of VOCALExplore's tasks results in substantial latency as we discuss in this section (and show in Section 5.4). The goal of the Task Scheduler is to optimize that latency without compromising the model quality seen by the user whenever they make API calls.

The Task Scheduler achieves this by making non-critical tasks asynchronous and performing just-in-time model training (Section 5.3.1), and by eagerly performing feature extraction while the user is occupied labeling (Section 5.3.2). These optimizations systematically target the principal sources of user-perceived latency.

Background. VOCALExplore has five types of tasks: feature extraction (T_f), model training (T_m), model inference (T_i), feature evaluation (T_e), and sample selection (T_s). Each EXPLORE call corresponds to multiple tasks of multiple types: VOCALExplore must first select a batch of video segments for labeling (this represents one task T_s per sample); extract features from the sampled segments if not already available (one task T_f per sampled video segment); perform inference with the latest model (one task T_i per sampled video segment); collect the labels from the user; train a new model (T_m); and evaluate feature quality for remaining features (one task T_e per feature; see Section 5.2.2). Additionally, if VOCALExplore needs to sample video segments using active learning instead of random sampling, it needs to sample more video segments than the user-requested number and extract features from the extra samples before selecting segments to return to the user for labeling, requiring a larger number of T_f tasks.

Baseline. Let T_{serial} be the API latency of a call to EXPLORE with a serial schedule, k the number of features still under consideration, and B the number of video segments labeled each iteration. With some abuse of notation, let's consider each T_x to represent not just the type of task but also the time to execute one such task. We then have: $T_{serial}^{random} = B(T_s+T_f+T_i)+T_m+kT_e$ for random sampling and $T_{serial}^{active} = (B+X)T_f+B(T_s+T_i)+T_m+kT_e$

when using active learning, where X is the number of extra samples that the ALM uses for active learning (Section 5.2.1). There are still only B T_s tasks because we only must select B samples (e.g., in CORESET, we perform B max-distance calculations).

The Task Scheduler does not minimize T_{serial} directly. Rather, we observe that the user spends a non-negligible amount of time, T_{user} to label each video segment. Given B video segments, after each call to EXPLORE, the user spends BT_{user} time labeling. The Task Scheduler exploits that time to do useful work in preparation for the next call to EXPLORE.

Problem statement. Let T_{total} be the time needed for VOCALExplore to return video segments to a user in response to a call to EXPLORE plus the time for the user to label the returned B video segments, so it represents the total time elapsed during a labeling session. The user returns labels $L_1, \ldots, L_B$. Given a sequence of calls to EXPLORE, the goal of the Task Scheduler is to minimize, at each iteration u, the user-perceived latency defined as: $T^u_{visible} = T^u_{total} - BT_{user}$, subject to maintaining good model quality. For the latter, given Q^u_{serial} the model quality (measured by any metric; we use macro F1 score) seen by the user for a serial schedule at iteration u, and $Q^u_{optimized}$ the model quality with the optimized schedule at the same iteration u, the Task Schedule seeks to ensure that $Q^u_{serial} - Q^u_{optimized} < \epsilon$. In our system, we do not start with a fixed ϵ but rather develop task scheduling approaches that empirically yield a small ϵ value.

We note that the prototype does not reduce visible latency to 0. Instead, it reduces it to $T^u_{visible} = B(T_s + T_i)$. However, it could be reduced further with speculative execution (i.e., prepare T_s and T_i before the next call to EXPLORE). We chose not to implement this in the current version of the prototype because T_s and T_i are small.

5.3.1 VE-PARTIAL *strategy*

Our first step towards an optimized strategy, VE-PARTIAL, uses the insight that not all tasks are equally critical for providing a response to API calls. Only selecting video segments, T_s, extracting features from them if not already available, T_f, and performing model inference, T_i, are required to return from EXPLORE. VOCALExplore hides model training latency by

performing inference over the most recent model that has already been trained. Similarly, feature evaluation tasks do not block EXPLORE; VOCALExplore updates the set of candidate features in the background as T_e tasks complete. The VE-PARTIAL strategy makes model training (T_m) and feature evaluation (T_e) asynchronous tasks, which reduces the user-perceived API latency to $T^{random}_{VE-partial} = B(T_s + T_f + T_i)$, and $T^{active}_{VE-partial} = (B+X)T_f + B(T_s + T_i)$. The quality of predictions is $Q^{u-\delta}_{VE-partial}$, where δ indicates how stale the model is.

The challenge the Task Scheduler addresses is to ensure that $Q^{u-\delta}_{VE-partial}$ is close to the quality achieved with the serial schedule. Using a model trained many iterations ago ($\delta \gg 0$) will not suffice because its quality is too low. Scheduling a new model training task after each new label is also not desirable. While this approach ensures that $\delta \approx 0$, it results in a factor of B more model training tasks, which causes congestion in the task queue. This approach also wastes resources because many models will never be used; when $T_m < (B-1)T_{user}$, multiple model training tasks will be queued and finish during a single iteration, but the ALM will make predictions using just the latest one.

The Task Scheduler addresses this challenge using "just-in-time" model training to minimize δ while still avoiding user-visible latency due to model training. The ALM tracks user labeling time (T_{user}) and model training latency (T_m). The ALM schedules a model training task after receiving $\max(0, B - \lceil T_m/T_{user} \rceil)$ labels because this ensures the model will be ready for inference by iteration $u{+}1$. When $T_m < T_{user}$, the ALM schedules a training task while the user labels the last example (i.e., after receiving L_{B-1}), so it makes predictions using a model trained with all but one label. If model training takes longer than an entire exploration iteration ($B - \lceil T_m/T_{user} \rceil < 0$), then the ALM schedules a model training task while the user labels the first sample. This model will not be ready for inference by $u{+}1$, but it will be ready by $u{+}\lceil T_m/(BT_{user}) \rceil$.

The VE-PARTIAL strategy reduces latency by making low-priority tasks asynchronous, and it maximizes model quality by scheduling "just in time" model training based on observed latencies.

5.3.2 VE-FULL *strategy*

The VE-PARTIAL strategy still has non-negligible latency due to feature extraction. $T_f \gg T_i$ because feature extraction operates over encoded videos, which requires expensive preprocessing, while inference operates over already-extracted feature vectors.

This leads to the Task Scheduler's second optimization: eager feature extraction. Strategy VE-FULL eagerly schedules feature extraction tasks (T_{f^-}) for unlabeled videos whenever the task queue is empty. These tasks have the lowest priority, so if any other task is scheduled while T_{f^-} is still queued, it will execute first. T_{f^-} tasks perform the same work as T_f, just at a lower priority. Initially, there are no unlabeled video segments from V with features extracted: $S{=}\emptyset$. The ALM randomly samples a set s of unlabeled video segments and schedules feature extraction tasks for all current candidate features, which results in a total of $k{\cdot}s$ T_{f^-} tasks. When these tasks complete, $S \leftarrow S \cup s$. The prototype sets $|s|{=}10$ to amortize the cost of setting up a feature extraction pipeline across multiple video segments while still completing the task within a few seconds.

The VE-FULL strategy has user-visible latency $T_{VE-full} = B(T_s + T_i)$ for both random sampling and active learning because the ALM uses S for both to eliminate feature extraction latency T_f.

Quickly converging to a single feature (Section 5.2.2) enables the most efficient growth of S because the number of T_{f^-} tasks is proportional to the number of candidate features. Growing S without affecting visible latency is desirable because it enables better active learning performance (shown in Section 5.4), and enables the user to get predictions over their dataset with minimal latency once they have a model they are happy with. It still is in the spirit of "pay-as-you-go" because the extra processing only happens while the user is interacting with the system. However, extracting features is expensive, and the user may not want to waste resources (e.g., by paying for a cloud GPU longer than necessary). Therefore,

VOCALExplore could add guardrails to stop eager feature extraction after some number of steps, either because the model quality plateaus or based on some heuristic of stopping once some fraction of the total dataset is processed.

5.4 Evaluation

We perform an evaluation of VOCALExplore. First, we show that compared to baselines, VOCALExplore achieves a high F1 score with the lowest latency, even while automatically performing feature and acquisition function selection (Section 5.4.1). Second, we demonstrate the effectiveness of the ALM's acquisition function selection process (Section 5.4.2). Then we demonstrate that the ALM's feature selection algorithm picks one of the best features within a small number of steps (Section 5.4.3). Finally, we evaluate the effectiveness of the Task Scheduler to show that it ensures low latency without hurting the F1 score (Section 5.4.4).

Implementation details. The prototype implementation is built using Python 3.8.10. The storage manager stores video metadata, labels, and model metadata in DuckDB 0.5.1 [122]. It stores feature vectors in Parquet files, and it uses PyTorch's [120] checkpoint capabilities to save models to disk. It uses the filesystem to store and retrieve encoded video files. Encoded video files are stored on hard drives, while all other data is stored on the local SSD. The feature manager uses NVIDIA DALI [10] to accelerate video decoding and to preprocess inputs to pretrained models when a GPU is available, otherwise it uses PyTorchVideo [49]. In the evaluation we perform feature extraction on the GPU, and the model manager trains and predicts using linear models.

Evaluation setup. We conduct all experiments on a compute cluster. When measuring runtimes, we request one node with eight Intel Xeon Gold 6230R CPUs @ 2.10GHz, 61GB of RAM, and one NVIDIA A40 GPU. This setup was chosen to approximate the memory, CPU, and GPU setup of a "p3.2xlarge" EC2 instance on AWS.

Datasets. We evaluate VOCALExplore on the datasets shown in Table 5.2. First, we evaluate on the Deer dataset which contains 10-second video clips captured from a camera attached to a collar on a deer [45]. We use a subset of the full dataset that we manually

Table 5.2: Datasets used to evaluate VOCALExplore

Dataset	# classes	Skew	Train videos	Eval videos
DEER	9	Skewed	896	225
K20	20	Uniform	13326	976
K20 (SKEW)	20	Skewed	1050	976
CHARADES	33	Skewed	7985	1863
BEARS	2	Uniform	2410	722
BDD	6	Skewed	800	200

labeled, which covers one day for a single deer. These clips show six activities that occasionally co-occur: bedded, chewing, foraging, grooming, looking around, and traveling. The activities are highly skewed towards the "bedded" activity. We create 5 train/eval splits by ordering the video clips temporally and taking every fifth one to be in the test set. Results are averaged across these splits.

We also evaluate on subsets of Kinetics700 [139], which is a standard video dataset comprising 700 human action classes. K20 contains 10-second video clips showing activities from 20 classes taken from the Kinetics700 dataset. We pick classes that do not appear in Kinetics400 to avoid overestimating performance for features that are extracted from models pretrained on Kinetics400. K20 is not skewed, however we introduce skew to create K20 (SKEW). The classes in the skewed dataset follow a Zipfian distribution with s=2. The most common activity has 650 videos and the least common activity has 3 videos. We create 10 training instances of K20 (SKEW) by permuting the classes. Results are averaged across these 10 instances. We use videos from the Kinetics validation set for evaluation, which is not skewed (even for K20 (SKEW)).

CHARADES [138] consists of 30-second videos showing 157 distinct activities. For our experiments, we simplify the task to identifying which of the 33 verb categories appear in each video.

Table 5.3: Features used by VOCALExplore. Throughput is the number of 10-second videos that can be processed each second while running two extraction tasks on the GPU.

Feature	Type	Architecture	Pretrained	Tput.
R3D [145]	Video	Conv. net	Kinetics400	4.03
MViT [50]	Video	Transformer	Kinetics400	2.93
CLIP [123]	Image	Transformer	Internet images	3.64
CLIP (Pooled) [123]	Image	Transformer	Internet images	3.45
Random	Video	Transformer	None	2.96

The Bears dataset consists of 5-second video clips captured from 19 camera traps in Alaska, primarily at night. The task is to determine whether or not each video clip contains a bear.

Finally, the BDD dataset [164] consists of 40-second video clips captured from moving cars. We extracted object detections from 1 fps using a Faster R-CNN model [125], and the task is to determine which objects (car, truck, person, bus, bicycle, and/or motorcycle) the 1.5 seconds covered by each feature vector contains.

Feature extractors. We initialize VOCALExplore with five candidate feature extractors shown in Table 5.3. We pick these feature extractors to cover image- and video-based models with a variety of architectures. For all of the features with input type "video", we use a sequence length of 16 (number of frames fed into the model), a stride of 2 (gap between frames in the sequence), and a step of 32 (gap between sequences). For the CLIP feature, we sample the middle frame out of every 32 frames so the feature aligns with the middle of the video feature windows. For the CLIP (Pooled) feature, we apply the CLIP model to every other frame from a window of 32 frames and perform max-pooling over the frame-level features. All of the features have 512 dimensions, except for MViT and Random which have 768 dimensions. We include the Random feature (which uses the same architecture as MViT but with randomized weights) to show that VOCALExplore handles low-signal features correctly.

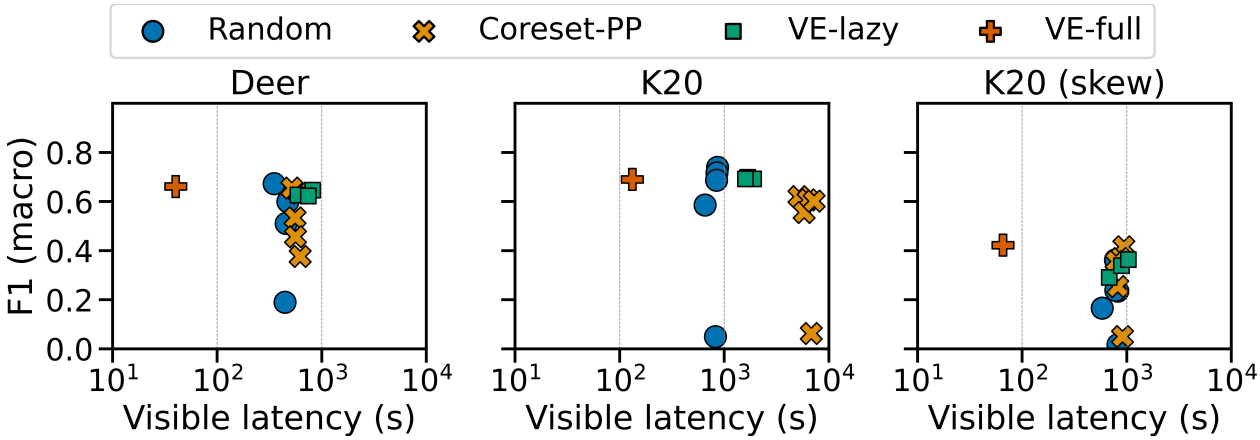


Figure 5.2: Average F1 and cumulative visible latency (shown with a log-scale) after 100 EXPLORE steps. CORESET-PP includes the preprocessing time to extract each feature, and each point for CORESET-PP and RANDOM represents a single feature. VE-FULL provides nearly the best model quality with the lowest visible latency.

Metrics. We evaluate model performance using macro F1 score because it is a standard evaluation metric. The F1 score is computed over the held out evaluation set after training a model on the labels collected so far at each step. We initialize VOCALExplore with the entire vocabulary that exists in the evaluation set so that it trains models that predict all evaluation classes, even when some classes don't have labels yet. We evaluate latency by measuring the wall clock time taken for VOCALExplore's API calls to return.

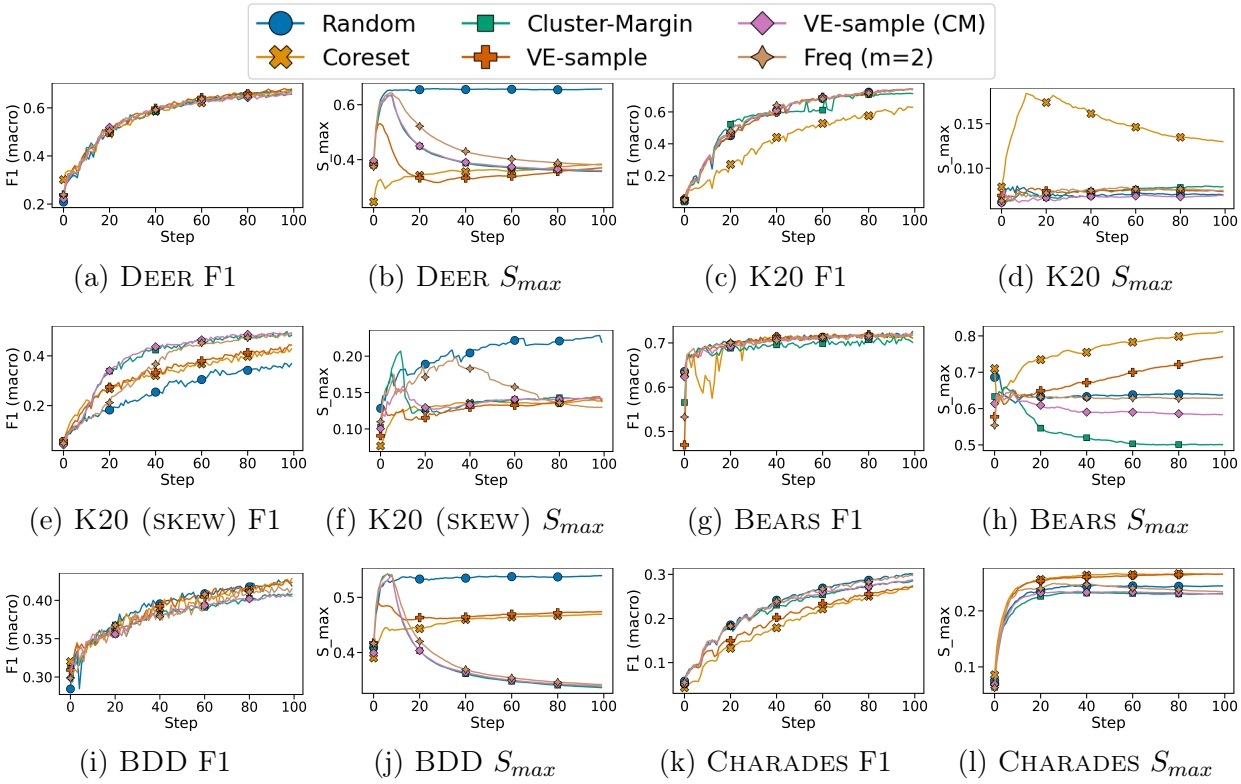
For the experiments below, we simulate a labeling task by creating an oracle "user" that labels video segments with their ground-truth labels. Labeling proceeds in a sequence of steps where we add five 1-second labels (which corresponds to EXPLORE(B=5, t=1)).

5.4.1 End-to-end performance

We first demonstrate that VOCALExplore achieves the best balance between visible latency and F1, as shown in Figure 5.2 (note that latency is shown with a log-scale). This experiment executes 100 calls to EXPLORE as described above. We measure the cumulative visible latency across these calls. RANDOM and CORESET-PP use the serial scheduler. RANDOM performs

random sampling over the videos, and we include a point for each candidate feature. All of RANDOM's latency comes from making predictions over the video clips returned from EXPLORE because its sampling latency is negligible. CORESET-PP uses CORESET sampling to select videos, and we include a point for each candidate feature. The cumulative latency includes the time it takes to extract each feature from all of the videos as a preprocessing step. VE-LAZY performs acquisition function and feature selection as described in Section 5.2, but without the scheduling optimizations described in Section 5.3. VE-LAZY incrementally extracts features from X additional videos if needed for active learning, as described in Section 5.3. The graphs show a point for each of $X \in [10, 50, 100]$. VE-FULL includes all of the scheduling optimizations described in Section 5.3. This experiment simulates the user taking 10 seconds to label each video clip, which is time VE-FULL uses to perform feature evaluation, train models, and eagerly extract features from videos. VE-FULL does not specify X; when the ALM switches to active learning it uses the features that have been eagerly extracted.

VE-FULL's model performance matches or exceeds VE-LAZY with a fraction of the visible latency, and its performance is close to the performance achieved by the best combination of acquisition function and feature. VE-FULL beats the model performance of VE-LAZY on K20 (SKEW) because VE-LAZY performs CORESET over a small sample of videos ($X \in [10, 50, 100]$), while VE-FULL extracts features from more videos in the background, and CORESET performs better over this larger sample. On the uniform K20 dataset, VE-LAZY has more latency than RANDOM because it performs feature evaluation. We discuss why the model quality of VE-FULL is lower than the best RANDOM point for K20 in Section 5.4.3. While CORESET-PP has higher visible latency than RANDOM, the difference is less on DEER and K20 (SKEW) than K20 for two main reasons. First, there are fewer total videos, so there is a smaller difference between the number of videos processed during the 100 EXPLORE steps and the number of videos processed during preprocessing. Second, there is overhead to creating each DALI feature extraction pipeline, so preprocessing all videos at once is more efficient because it can use a single pipeline.



(a) DEER F1 (b) DEER S_{max} (c) K20 F1 (d) K20 S_{max}

(e) K20 (SKEW) F1 (f) K20 (SKEW) S_{max} (g) BEARS F1 (h) BEARS S_{max}

(i) BDD F1 (j) BDD S_{max} (k) CHARADES F1 (l) CHARADES S_{max}

Figure 5.3: VOCALExplore's data sampling method yields models with the highest F1 scores and samples from a diverse set of classes (S_{max}, lower is better) across datasets with different degrees of skew.

The optimizations from Section 5.3 could be applied to RANDOM and CORESET-PP to reduce their latency, however that does not solve the problem of how to pick the correct combination of acquisition function and feature for an arbitrary dataset. As shown in Figure 5.2, model quality differs significantly across combinations.

5.4.2 Acquisition function selection

We now focus on the effectiveness of the ALM's acquisition function selection, as discussed in Section 5.2.1. We compare against baselines of using a fixed function: either always performing RANDOM, CORESET [133], or CLUSTER-MARGIN [36] sampling. VE-SAMPLE

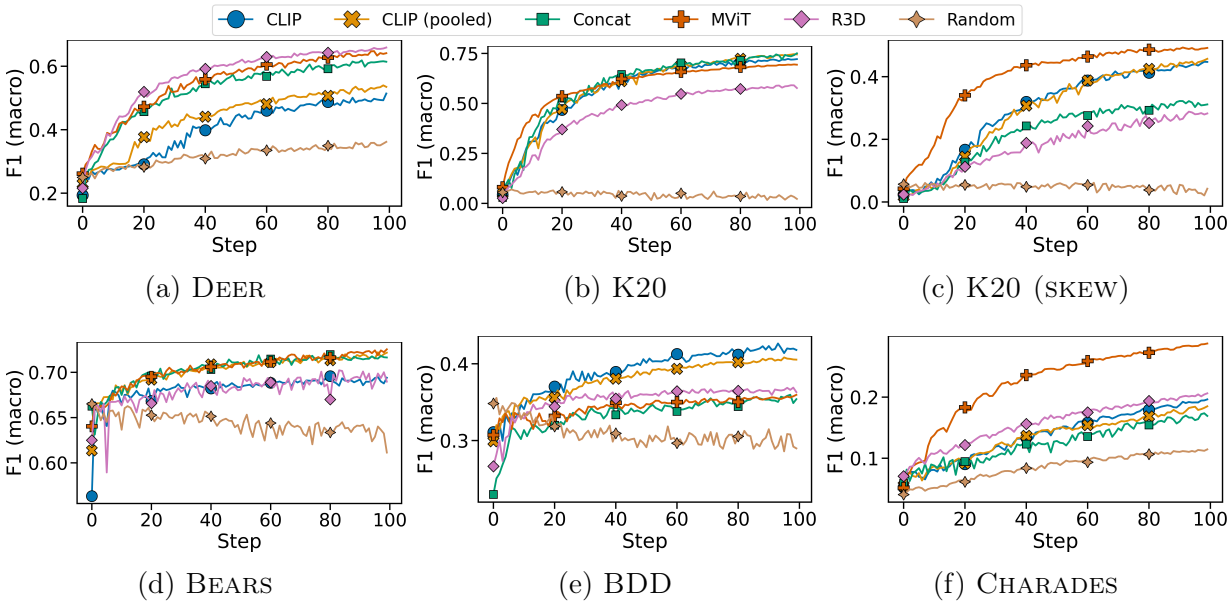
picks between RANDOM and CORESET at each iteration as described in Section 5.2.1, while VE-SAMPLE (CM) picks between RANDOM and CLUSTER-MARGIN. FREQ. also picks between RANDOM and CLUSTER-MARGIN but uses the frequency-based test described in Section 5.2.1. For this experiment, we show results only for the best feature (Figure 5.4). We evaluate with R3D for DEER, MViT for K20 (SKEW) and CHARADES, and CLIP (POOLED) for K20, BEARS, and BDD.

We measure performance by both the macro F1 score of the model, as well as a diversity metric S_{max}, which computes the fraction of labels that come from the single most-seen activity (see Section 5.2.1). A smaller S_{max} indicates that the user sees more diverse examples, which makes the labeling task more interesting.

First, Figure 5.3 shows that CLUSTER-MARGIN (and therefore VE-SAMPLE (CM)) always perform at least as well as CORESET and VE-SAMPLE. Therefore, we limit the rest of our discussion to RANDOM, CLUSTER-MARGIN, and VE-SAMPLE (CM).

Looking at the uniform datasets of K20 and BEARS, we observe that RANDOM produces models with the same F1 score as CLUSTER-MARGIN. Therefore, it is unnecessary to sample these datasets with the more expensive active learning technique. Looking at the skewed datasets, we observe that using active learning boosts the F1 score above RANDOM for K20 (SKEW). We also see improved (i.e., lower) S_{max} metrics for the skewed datasets when using CLUSTER-MARGIN. Therefore, it is useful to use active learning on skewed data because it is possible the model performance will be improved, and the user is likely to see a more diverse set of examples to label. We observe that VE-SAMPLE (CM) matches the performance of the best technique on each dataset by detecting whether the labels are skewed and switching to active learning if appropriate.

Finally, we observe that using the frequency-based method for determining whether a dataset is skewed leads to similar results as the Anderson-Darling k-sample test, though it is slightly more conservative and takes longer to switch to an active learning sampling method. This can be modified by adjusting m; we don't show the results to avoid crowding the graphs, but using m=1.5 leads to curves that more closely match VE-SAMPLE (CM).



(a) DEER (b) K20 (c) K20 (SKEW)

(d) BEARS (e) BDD (f) CHARADES

Figure 5.4: Macro F1 score when using the VE-SAMPLE (CM) sampling method, which shows that the best feature varies across datasets. "Concat" refers to concatenating all of the features into a single feature vector.

Table 5.4: Feature selection correctness.

	DEER	K20	K20 (SKEW)	BEARS	BDD	CHARADES
$T = 20$	1.00	1.00	0.98	0.97	0.50	0.87
$T = 50$	0.99	1.00	1.00	0.95	0.69	0.92

5.4.3 Feature selection

We now evaluate the effectiveness of the ALM's feature selection algorithm. We measure the correctness (i.e., how *frequently* do we pick one of the best features) and the efficiency of the selection (i.e., how *quickly* do we pick a feature). We initialize VOCALExplore with the five candidate feature extractors from Table 5.3.

We first evaluate the correctness of feature selection. To measure the quality of each feature, in Figure 5.4, we compute the macro F1 score for each feature across 100 labeling iterations (using VE-SAMPLE (CM) to pick video segments). It includes CONCAT to show

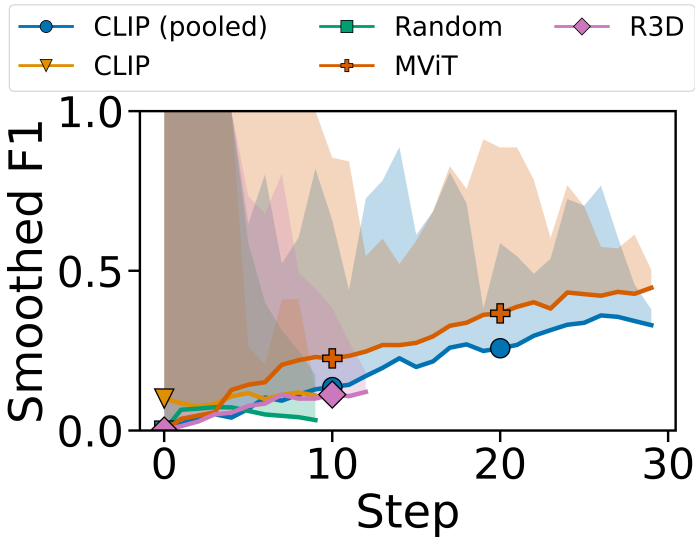
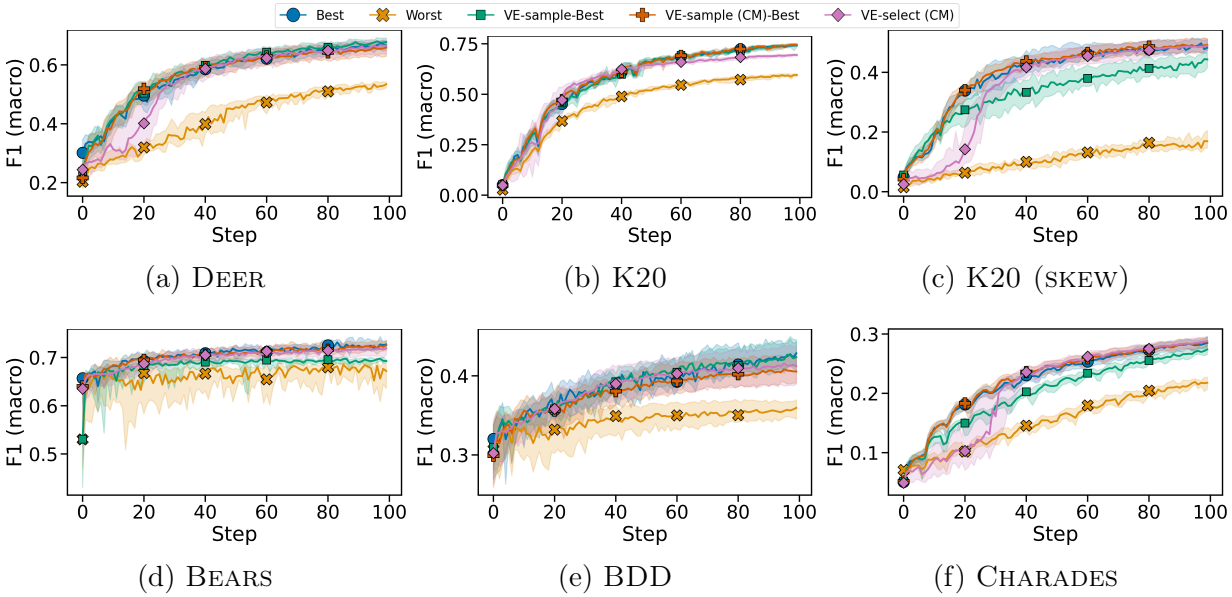


Figure 5.5: Feature selection progress for K20 showing upper and lower bounds when C=5, w=5, T=50.

that concatenating all of the potential features does not improve performance over the best single feature. Based on these results, we use the following rules when determining the correctness of feature selection. For the DEER dataset, we consider selecting either R3D or MVIT to be a correct decision. For K20 and BEARS, we consider any of MVIT, CLIP, or CLIP (POOLED) to be a correct decision. For K20 (SKEW) and CHARADES we consider only MVIT to be correct. For BDD we consider CLIP or CLIP (POOLED) to be correct. In this experiment we use C=5, w=5. We discuss sensitivity to hyperparameter values at the end of this section.

Table 5.4 shows that the ALM picks a correct feature at least 92% of the time (excluding BDD) when the time horizon is long enough (T=50). When the algorithm picks incorrectly, it primarily picks the next-best feature (e.g., one of the CLIP features for DEER or K20 (SKEW)). The algorithm selects incorrect features for BDD some of the time because all features perform similarly until later iterations when CLIP and CLIP (POOLED) start to perform better. Therefore, despite the correctness measure being low, the F1 score achieved is close to the best as shown in Figure 5.6e. The algorithm struggles with CHARADES due to



(a) DEER (b) K20 (c) K20 (SKEW)

(d) BEARS (e) BDD (f) CHARADES

Figure 5.6: Macro F1 score when performing feature selection compared to the empirically best- and worst-performing sampling methods and features (excluding the RANDOM feature). We also compare against VE-SAMPLE and VE-SAMPLE (CM) sampling methods on the best feature. VOCALExplore initially has poor F1 performance as it explores suboptimal features but catches up to the best strategies within 30 steps. The shaded region shows the IQR.

the noise introduced by evaluating with k-fold over the large number of classes; correctness is $\geq$95% when evaluating with the full test set as described at the end of this section. The performance over CHARADES with k-fold can be improved to 98% correct by using stronger smoothing (w=7, C=7).

?? shows that the ALM picks a single feature within a small number of iterations. Convergence is faster when T=20 than T=50 because the upper bounds on the expected performance have lower values, so features are eliminated more quickly. Even at T=50, features are selected within about 30 steps. Figure 5.5 shows an example of how the upper and lower bounds evolve for K20. We use T=50 in the rest of the experiments.

We also evaluate the model quality as the ALM performs feature selection (VE-SELECT). Figure 5.6 shows that while VOCALExplore initially has sub-optimal model quality as it explores features, it catches up to the best-performing strategies within approximately 30 steps. We compare against BEST and WORST, which correspond to the empirically best- and worst-performing combinations of sampling methods and features (excluding the RANDOM feature) to show the range of expected values. We also compare against VE-SAMPLE and VE-SAMPLE (CM) on the best feature (VE-SAMPLE-BEST and VE-SAMPLE (CM)-BEST, respectively). We observe that initially VE-SELECT's performance is close to the worst-performing strategy because it has poor-performing features as candidates. These features result in models with low F1 scores, and they also hurt CORESET sampling because distances in their feature spaces are not meaningful. The VE-SELECT curve exhibits an "S" shape, where once it converges to a single feature, performance catches up to the best-performing strategies. While K20 does not converge to a single feature until 30 steps, the model quality improves before then because the bad features are eliminated, and all remaining candidates perform well. K20's final model quality is slightly lower than the best because it picks MVIT 98% of the time, and MVIT has the highest quality when there are few labels but not when there are a larger number of labels (as shown in Figure 5.4b). This illustrates that because we use a small T value to encourage quick convergence to one feature, the ALM's feature selection algorithm is biased towards features that perform well in early iterations.

Finally, we evaluate the sensitivity of the hyperparameters. We perform this analysis when measuring quality using the evaluation set rather than performing 3-fold validation over the labeled set in order to evaluate the behavior of feature selection under more ideal settings. We find that the quality is $\geq$95% for all datasets except BDD across a reasonable range of hyperparameter values ($w \in [3, 5, 7], C \in [5, 7], T \in [20, 50]$). BDD's selection correctness ranges from 0.68 to 0.88 for all settings. The evaluation set gives a more reliable estimate of feature quality, so the correct feature is picked even with less smoothing and a shorter time horizon.

5.4.4 Task scheduler

Finally, we evaluate the effectiveness of the optimizations described in Section 5.3 and show they enable VOCALExplore to match or exceed the model quality of VE-LAZY but at a fraction of the visible latency. Figure 5.7 shows model quality and cumulative visible latency across 100 EXPLORE steps (note that latency is shown with a log-scale). As in Section 5.4.1, we assume the user takes 10 seconds to watch and label each video clip. The VE-LAZY variants perform feature and acquisition function selection as described in Section 5.2, but without the optimizations from Section 5.3. VE-LAZY (PP) includes the preprocessing time to extract *all* candidate features from all videos, which is necessary because the ALM does not initially know the best feature. VE-LAZY (X) variants perform incremental feature extraction as needed when the ALM switches to CORESET sampling. X indicates the number of unlabeled videos that have features extracted to serve as the candidates for CORESET. Larger X values have higher F1 on K20 (SKEW), and to a lesser extent on DEER, but the additional feature extraction tasks increase visible latency. Finally, VE-FULL, which uses all of the optimizations described in Section 5.3 matches or exceeds the F1 score achieved by the lazy variants but at much smaller visible latency (which ends up being $\sim$1 second per step). VE-FULL exceeds the performance of the lazy, incremental variants on K20 (SKEW) because it extracts features from more videos in the background than the values of X we evaluated, so CORESET sampling performs better.

5.4.5 Label quality

In this section we evaluate the impact of noisy labels on VOCALExplore's performance. We use the same experimental setup as in Section 5.4.3 and initialize VOCALExplore with the five candidate feature extractors from Table 5.3. We perform 100 labeling iterations and

measure the F1 score as the ALM performs feature selection. As in Figure 5.6, we compare against the best- and worst- performing combinations of features and sampling methods as well as VOCALExplore's performance when there is no label noise.

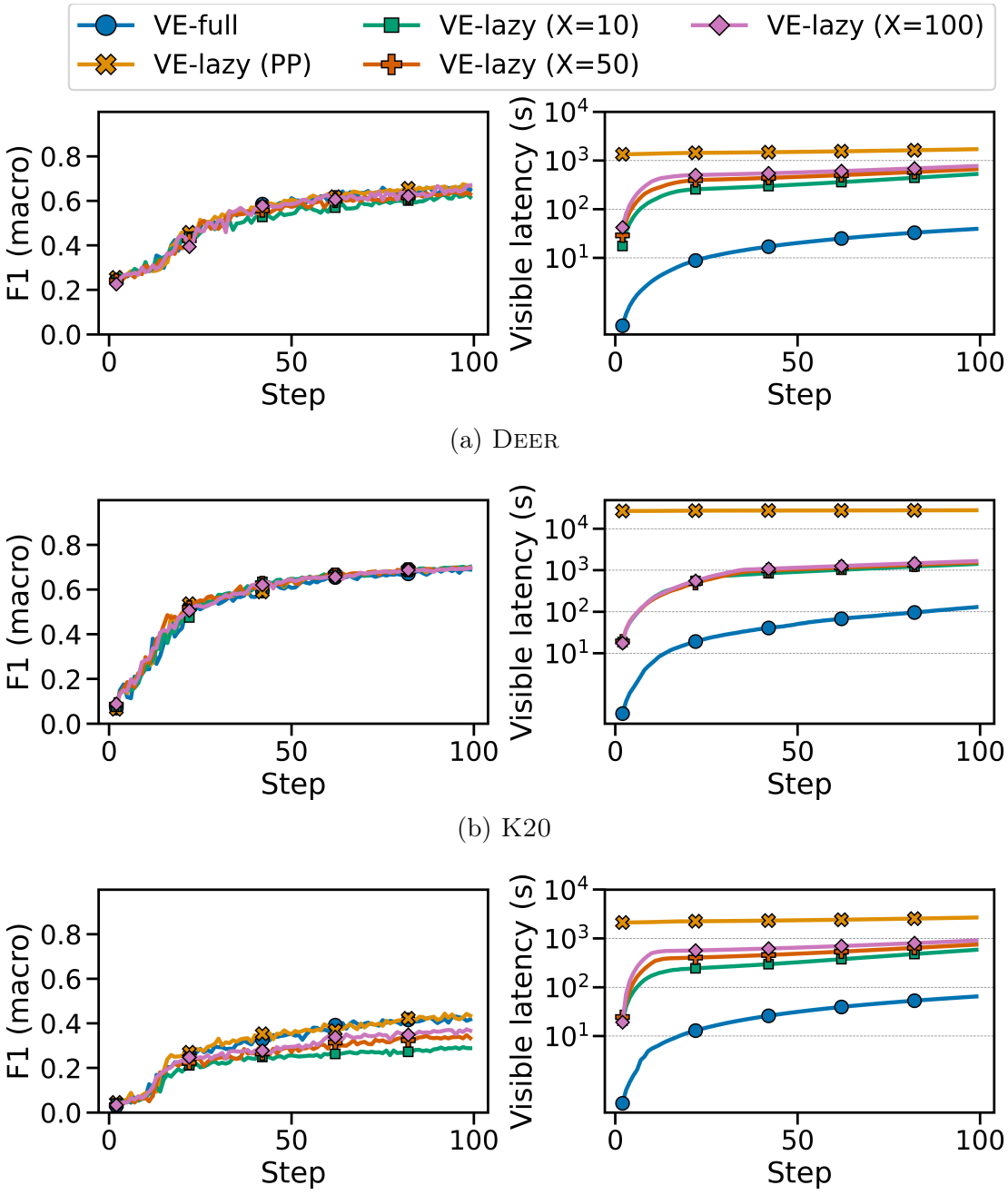
We use a noisy oracle to label the selected video segments that randomly changes labels 5%, 10% or 20% of the time. As shown in Figure 5.8, the performance with 5% and 10% noise is close to the performance with no noise. There is a drop in performance with 20% noise, but the F1 score is still better than the worst-performing feature and sampling method. This indicates that the techniques the ALM uses to select acquisition functions and features are robust to reasonable amounts of noise.

5.5 Summary

This chapter presented VOCALExplore, a system that supports exploring and building domain-specific models over videos. VOCALExplore does so in a pay-as-you-go manner, so the user does not need to wait for an expensive preprocessing step before interacting with the system. The system builds an increasingly-accurate model as the user spends more time interacting with it and labeling more data. VOCALExplore does not presume users have any ML expertise; it automatically determines how to select samples to be labeled and picks the best feature extractor for a given dataset. To maintain interactivity, it implements optimizations to minimize the latency of API calls while maintaining model quality.

We evaluate VOCALExplore on domain-specific wildlife datasets showing deer and bears, as well as on standard activity classification and self-driving car datasets. We show that VOCALExplore can achieve model performance that matches the best combination of sampling function and feature extractor with no preprocessing and a user-visible latency of less than one second per labeling iteration.

VOCALExplore fills an important usability gap that currently keeps many users with video datasets from benefiting from the data management and query processing capabilities of VDBMSs. Users whose datasets lack an off-the-shelf model to extract semantic information, which is required by current VDBMSs, can utilize VOCALExplore to efficiently build such a model.

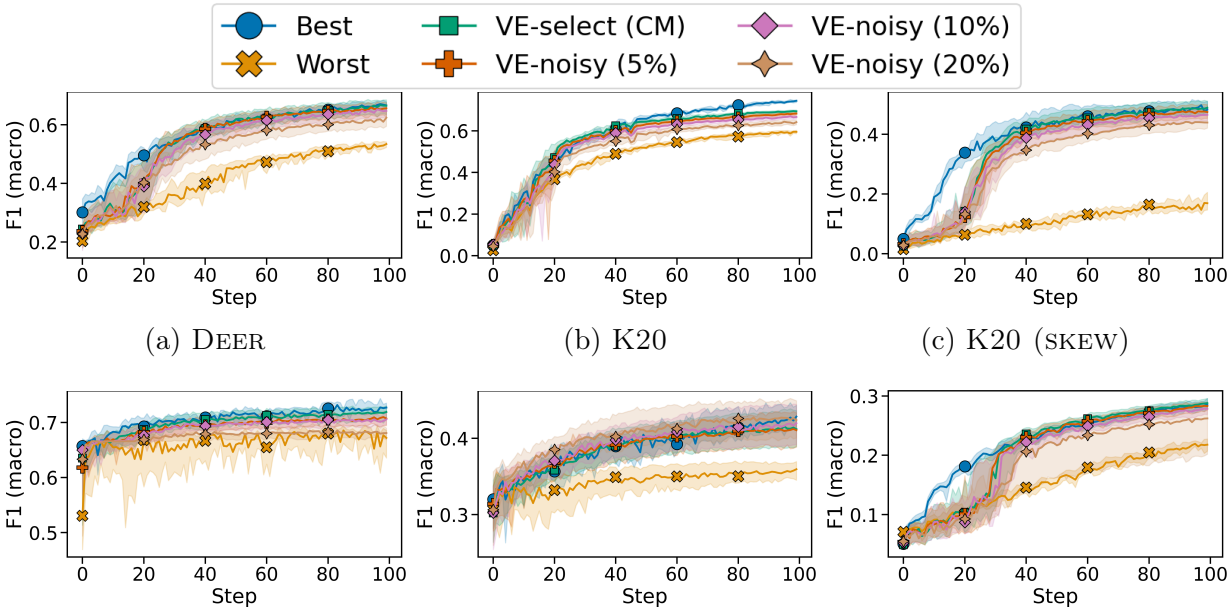


(a) DEER

(b) K20

(c) K20 (SKEW)

Figure 5.7: Model quality and latency for VE-variants. VE-FULL matches the best model performance of VE-LAZY with less cumulative visible latency (shown with a log-scale).



(a) Deer (b) K20 (c) K20 (skew)

(d) Bears (e) BDD (f) Charades

Figure 5.8: Macro F1 score when performing feature selection with noisy labels (VE-noisy) compared with no noise (VE-select (CM)) and the empirically best- and worst-performing sampling methods and features (excluding the Random feature). We evaluate performance when randomly changing 5%, 10%, and 20% of labels.

Chapter 6

VOCALEXPLORE: INTERFACE AND EXTENSIONS

The previous chapter described the components and algorithms underlying VOCALExplore, which implements an API to enable interactive exploration and domain-specific model building over videos. This chapter starts by presenting VOCALExplore's user interface which builds on top of this API and facilitates labeling videos in addition to searching over labels and model predictions (Section 6.1). We then describe how VOCALExplore could be extended to support additional labeling tasks and more fine-grained label control (Section 6.2) and incorporate multimodal features, e.g., from the audio tracks of videos (Section 6.3).

6.1 VOCALExplore interface

As discussed in Chapter 5, video data is increasingly common in scientific and engineering domains, and novel video database management systems (VDBMSs) have been developed to support the needs of modern video applications [160, 113, 81]. Many users, however, are unable to benefit from these recent advances because current systems assume users already have a machine learning (ML) model that can be used to extract semantic information from the videos. Many domains, however, lack such pretrained models.

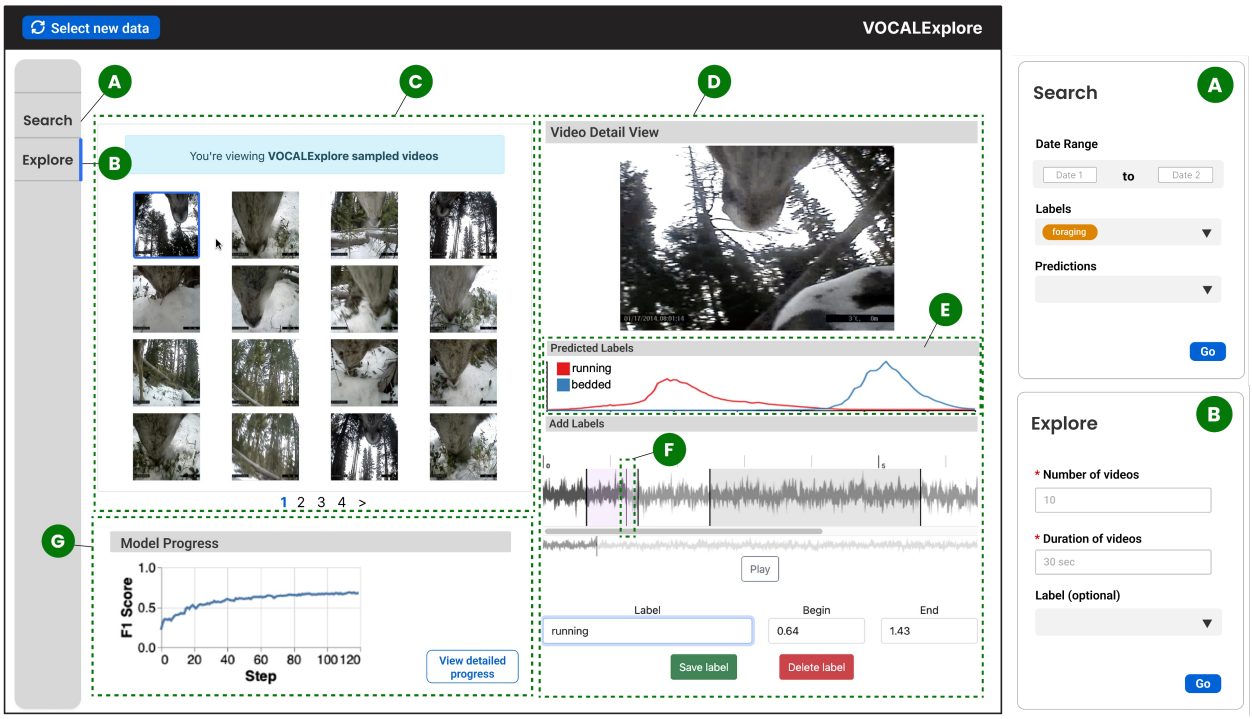
To briefly summarize the previous chapter, VOCALExplore is a pay-as-you-go system for video data exploration and domain-specific model building that addresses the above challenge. VOCALExplore takes as input a collection of videos. It guides users through efficient data sampling and labeling to produce a high-quality model for a new domain, which users can then apply to annotate their entire video collection. VOCALExplore builds on techniques from the ML literature and makes three key contributions: (1) it brings together state-of-the-art ML methods and exposes them through a data exploration API that does

not require any ML knowledge from users; (2) it automatically and dynamically adapts to each new dataset in order to produce high-quality models with small numbers of user labels. Specifically, it (2a) changes sample acquisition functions (i.e., between random and active learning) based on data distribution; and (2b) simultaneously dynamically explores and picks the best feature extractor for a given set of videos. Finally, (3) VOCALExplore includes a task scheduler that ensures low-latency interactions, while maintaining high model quality.

VOCALExplore provides an API with the basic functions needed to explore and label a video dataset. However, this alone is not sufficient to support users. Videos are an inherently visual medium, so a graphical interface is necessary to enable users to navigate and annotate their datasets. This section introduces VOCALExplore's user interface, which is built on top of the API described in Chapter 5, to facilitate user interactions necessary for exploration and domain-specific model building. The user interface supports manual exploration via SEARCH, where users can search over video metadata (e.g., date), labels they have applied, or predicted labels generated by VOCALExplore. The interface also supports system-driven exploration via EXPLORE, where a user can specify their labeling budget, the duration of videos they want to see, and optionally a specific class they want to focus labeling on. VOCALExplore then populates the interface with unlabeled video segments that are expected to most improve model performance after the user labels them.

With its graphical interface, VOCALExplore supports interactive, pay-as-you-go video data exploration. It enables users to load raw videos and figure out the labels they want to apply as they explore their data; the system supports these users in finding, labeling, and ultimately automatically annotating video segments. It does this in a non-blocking and interactive manner.

VOCALExplore's interface enables a user to view videos and associated predictions, and to add labels. Because labeling videos is a tedious task, it is important to have an interface that reduces friction for common operations like adding labels and minimizes latency to enable interactivity. We introduce the components of the interface by walking through an example use case.



(a) Main screen (b) Detail views

Figure 6.1: (a) shows the user interface for VOCALExplore, and (b) shows detail views that are presented after clicking on Ⓐ or Ⓑ in the main screen.

Data ingestion and initial exploration. Users load a new dataset using the "Select new data" button at the top of the page (Figure 6.1a). We do not show this view in the thesis, but after the user enters the paths to their videos, they are shown the main screen.

The user can start with manually browsing through a few of their videos using the "Search" interface Ⓐ and specifying a desired date range. Later, they will also be able to "Search" based on labels, but there are initially no labels and no predictions. Videos satisfying the search predicate populate the *video grid view* Ⓒ, and the user can play selected videos in the *video detail view* Ⓓ. We describe these views in more detail below. This manual exploration

lets the user learn what their videos look like and verify that they match their expectation. Once the user has manually skimmed through a small number of videos, they are ready to begin labeling and training a domain-specific model.

VOCALExplore-driven exploration and labeling. After the user clicks on the "Explore" button in the sidebar, they are shown a short form Ⓑ where they fill in their labeling budget (i.e., how many video segments they are willing to label), the duration of video clips they want to view (i.e., some users may prefer to view a smaller number of long segments while others may prefer to see a larger number of short segments; the choice may also depend on the videos and application), and optionally a specific activity type they want to improve model performance on. Once they click the "Go" button, VOCALExplore populates the *video grid view* Ⓒ with video segments it selects on behalf of the user. The user interface limits the number of videos rendered at once to manage its computational overhead, so the user may have to page through the returned videos if their labeling budget is large.

When the user selects a video from the grid, the video appears in the *video detail view* Ⓓ. This view includes a video player for the user to watch the video. It also includes a *prediction view* Ⓔ that shows the predicted labels returned by VOCALExplore. This view is empty until the first round of user labels initializes VOCALExplore's training set, so it can start training models. VOCALExplore returns a set of $activity \rightarrow probability$ pairs, each associated with a time range. The interface displays these probabilities using a line graph aligned with the video timeline, with one line per activity. This lets the user jump to potentially interesting points in the video and verify whether the predictions are correct.

Beneath the prediction view is the *labeling view*. This view enables users to add or edit labels. VOCALExplore's interface builds on the WAVESURFER plugin [12]. The user drags to select a duration along the audio waveform, where the current playback location is shown by the vertical bar Ⓕ. The user can then provide one or more labels for the selected time range. When the user clicks the "Save Label" button, VOCALExplore stores these labels and updates the model it uses to make predictions on unlabeled videos.

The first time the user fetches videos with EXPLORE, no predictions are returned because VOCALExplore does not yet have any labels to train a model. However, future EXPLORE iterations populate the prediction view Ⓔ. After VOCALExplore trains an updated model, it proactively schedules inference tasks to generate and cache model predictions over videos. This makes populating the prediction view a low-latency operation. It also ensures that if the user executes a manual query over the model predictions, the system is already processing the inference tasks and updating the cache. Therefore, VOCALExplore can *immediately* return partial results over the predictions currently in the cache even if not all prediction tasks have completed by the time the user executes the search.

Strengths of VOCALExplore's current interface. As described in this section, VOCALExplore's interface is designed to support users in activity classification tasks. It provides elements for users to easily label video segments, view model predictions for videos, and search over labels, predictions, and other video metadata. However, the interface currently does not show statistics about model performance that would support users in reasoning about the model's strengths and weaknesses across classes. Additionally, neither the labeling interface nor the underlying system supports more fine-grained labeling tasks that require applying annotations to subregions of frames. Further, while users may change their mind about the labels they want to use or would like predictions from a model trained on a subset of labels, VOCALExplore does not provide affordances for this advanced label control.

6.2 Extensions to VOCALExplore

In this section we describe how VOCALExplore explore could be extended to convey additional information about model quality in its user interface, and how it could support alternative labeling tasks and fine-grained label control.

6.2.1 Model quality visualizations

While the prediction view from Figure 6.1a enables the user to assess model quality on individual videos by manually inspecting the predictions while watching the video, the user will also want to view aggregated model quality statistics. VOCALExplore could be extended to let the user track model training progress via a proposed *model progress* view (Figure 6.1a Ⓖ) which shows the progression of estimated model quality over time. VOCALExplore must estimate model quality using k-fold cross-validation because we do not assume the user has a labeled validation set. By default VOCALExplore could show just the macro performance averaged across all known labels. Users could view the performance plotted for each label by clicking the "View detailed progress" button. These two views would give the user insight into how the model is performing on the classes they care about. Based on these metrics, the user may begin specifying classes in the EXPLORE form (Figure 6.1b Ⓑ) to improve the model's accuracy on those classes. Alternatively, the user may be satisfied with the model performance and begin searching for specific instances of activities of interest based on the model's predictions. The user can do so by specifying an activity and probability range in the "Predictions" field of the search form (Figure 6.1b Ⓐ).

6.2.2 Supporting additional labeling tasks

VOCALExplore supports activity classification tasks where the input to the labeling task is a video segment, and the output is a set of predictions for possible classes that appear in the segment. However, there are many other labeling tasks that users perform over videos. For example, in activity localization the input is similarly a video segment, but the output is a set of predictions for possible activities in addition to their predicted start and end timestamps. Or, in object segmentation the input is a video frame, and the output is a mask assigning each pixel to a region (e.g., marking pixels as belonging to a particular object).

VOCALExplore could be extended to support additional labeling tasks. We describe the necessary changes in the context of object detection, where the goal is to identify and locate objects within frames by outputting bounding boxes with associated labels. To support object detection, VOCALExplore would have to solve two ML tasks: *region proposal* and *region classification*. To obtain region proposals, VOCALExplore would have to decide between using a pretrained region proposal model vs. training its own with supervised data. Pretrained region proposal models will not work well if the objects in the video frames exhibit different "objectness" properties from what the model was trained on (e.g., objects in histopathology images have different properties than objects in wildlife images). Therefore, VOCALExplore would need to implement a decision function to detect when the regions proposed by a pretrained model are not of sufficient quality. In these instances, VOCALExplore would have to train a domain-specific region proposal model in addition to a domain-specific classification model.

Once VOCALExplore can identify regions in frames that likely contain objects, then it becomes a classification task again. However, instead of extracting features from entire frames, VOCALExplore would have to extract features from just the regions of interest.

While adding additional feature extractors would not be required to support object detection, it may be beneficial. We expect that the best feature extractors to add would be models pretrained on object detection. Finally, the task scheduler would not need to be tuned because it dynamically schedules tasks based on observed latencies. Model training and user labeling latencies will likely change, but the task scheduler will adapt by scheduling tasks more- or less-eagerly.

6.2.3 Supporting fine-grained label control

The current prototype of VOCALExplore trains models over *all* labels the user has applied to video segments. However, users may change their mind about the set of labels they want to use (e.g., they may want to rename some particular label or apply an additional label to video segments already annotated with a particular class). Users may also want to obtain model

predictions for just a subset of labels (e.g., they may label videos with both activities and location but want separate models for each task, or, they may apply a hierarchy of labels to video segments and desire separate models over the coarse-grained labels vs. the fine-grained labels).

It would be straightforward to extend VOCALExplore to provide this type of label control. We will first consider users who want to change or add labels to video segments based on their current labels. VOCALExplore's underlying API supports adding, deleting, or modifying labels, so this simply requires adding affordances in the user interface for users to select multiple video segments and specify their desired label transformation. Currently the interface requires users modify labels by clicking on the timeline of a specific video, so it would be necessary to add a labeling interface separate from the video player.

We will now consider users who want to train models over subsets of labels and who want to control which model is used to make predictions. This extension requires modifications at both the API and the interface level. First, VOCALExplore's model training and prediction API calls must accept an additional parameter that indicates the set of labels that should be considered when training a model or picking a model to use for inference. Then, the user interface must make it possible for the user to specify these label sets when viewing or searching over predictions.

The extensions to VOCALExplore discussed in this subsection would give flexibility to users and enable them to build on and refine labels applied during early exploration rather than requiring them to engage in a tedious re-labeling process whenever the target task changes.

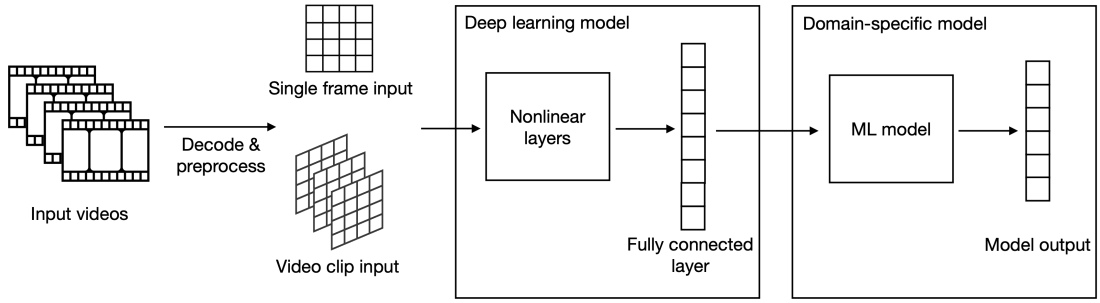
6.3 Extending VOCALExplore to utilize multimodal features

As described in Chapter 5, VOCALExplore supports video exploration and domain-specific model building by extracting feature vectors from video segments, and then using these features as input to models for training and making predictions. VOCALExplore performs this transformation from pixels in one or more frames to a feature vector using pretrained

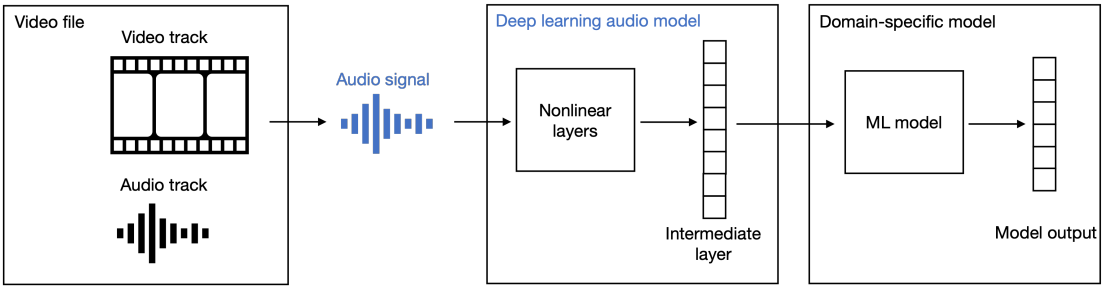
image and video models, as described in Section 2.2. The system does not use these pretrained models to predict the activities and objects that appear in the videos because it assumes that the dataset being analyzed comes from a different domain than what the models were trained on. Instead, VOCALExplore uses these models as feature extractors by retrieving an embedding from one of the model's intermediate layers.

While VOCALExplore currently only considers pretrained image and video models (Table 5.3) that extract features from the pixel data, videos are often accompanied by audio tracks that encode additional information about the video content. The audio data may be *complementary* to the video data; for example, if the video shows heavy traffic, the audio track may encode honking horns. In other settings, the audio data may actually be *more* informative than the video data. This may happen when the camera lens is obstructed or lighting conditions are poor, so the video does not capture the intended content. Finally, the audio data may be *uncorrelated* with the video data and not add any information useful for identifying what is happening in the video. For example, audio tracks from videos in the Deer dataset (Table 5.2) all consist of quiet rustling that is not useful for distinguishing activities. Similarly, many videos in the K20 dataset are set to an unrelated song that is ill-suited for determining the video content.

This section discusses how VOCALExplore could incorporate multimodal features, like those from audio tracks, to augment the domain-specific models it builds. We first describe the process of extracting feature representations from audio tracks (Section 6.3.1), and then we discuss the changes required for VOCALExplore to utilize these multimodal features (Section 6.3.2). We conclude by presenting initial experimental results (Section 6.3.3) and discussing future directions of investigation motivated by the initial results (Section 6.3.4).



(a) Using a pretrained model as a feature extractor for frames. The domain-specific ML model can have any architecture.



(b) Using a pretrained audio model as a feature extractor for audio tracks.

Figure 6.2: Model inference and feature extraction for audio inputs. (a) shows the pipeline for video frames, and is duplicated here from Figure 2.5b for convenience. (b) shows the pipeline for audio tracks extracted from videos. The differences are highlighted in blue: first, the input to the pretrained model is the audio signal rather than frames. Second, the pretrained model is specialized for audio rather than visual tasks and expects audio as input.

6.3.1 Background on generating audio features

The process of extracting features from the audio track of videos is very similar to how VOCALExplore extracts features from frames. Figure 6.2a shows the feature extraction process for visual features; it is copied here from Section 2.2 for convenience. To summarize Section 2.2, features are extracted from frames after decoding the video track to pixels. These pixels are preprocessed and passed as input to a pretrained image or video model.

Figure 6.2b shows the process to extract audio features. Rather than decode the video track from a video file, VOCALExplore decodes the audio track to produce a *waveform* that represents the audio signal over time. The waveform is resampled to the frequency expected by the pretrained model and then passed as input to this model. Rather than use an image or video model, VOCALExplore must use a pretrained audio model, such as Wav2vec 2.0 [19]. However, the output of audio models has the same format as the output of image and video models; both audio and visual features are represented by d-dimensional feature vectors, where the dimension d may vary across models.

6.3.2 Incorporating audio features into VOCALExplore

Once VOCALExplore extracts feature vectors from the audio tracks they can be used for model training and inference, just as the system currently uses feature vectors from the video tracks. However, there may be benefits to VOCALExplore considering the source of the features (i.e., audio vs. video). In this subsection, we discuss the design implications for how VOCALExplore can incorporate audio features into its feature selection and model training. While this section describes the minimal changes required for VOCALExplore to utilize multimodal features, a full treatment of this topic should evaluate against techniques previously proposed to optimally choose between and fuse features from multimodal channels [18, 154].

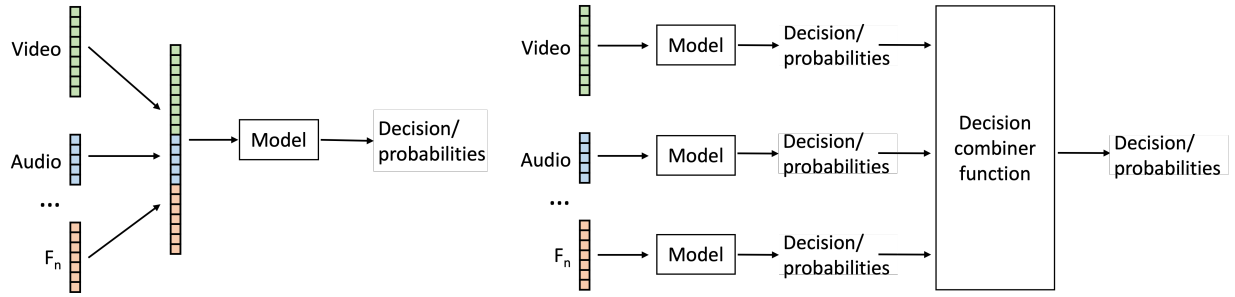
Feature selection. Recall from Chapter 5 that VOCALExplore's Active Learning Manager (ALM) is responsible for picking a feature extractor that produces high-quality models on the given dataset. The ALM picks a feature extractor from a set of candidates, which currently consists of image and video models (Table 5.3). The simplest way for the ALM to incorporate audio feature extractors would be to add them as additional candidates alongside the existing image and video options. The ALM would select a single feature extractor (as it does now) to train domain-specific models, so the audio features would only be used if they outperform the visual features.

To formalize this, if we let V be the set of all video feature extractors, and A be the set of all audio feature extractors, the naive approach is for the ALM to pick $\arg\max_{f \in V \cup A} M(f)$, where $M(f)$ is the model quality achieved using feature extractor f. As described in Section 5.2.2 the ALM cannot directly evaluate $M(f)$. Instead, the ALM would use its rising bandit algorithm over candidates $V \cup A$ to select one of the best feature extractors based on the empirical performance of each candidate over time.

While in most datasets it is unlikely that the audio features on their own encode *more* information than the visual features, they may encode information that *enhances* the visual features. This insight leads to the *multimodal strategy* where the ALM may select a combination of feature extractors from both the visual and audio domains for training downstream models. Specifically, whereas in the naive strategy the candidate feature extractors are $V \cup A$, in the multimodal strategy the candidates are $V \cup A \cup \{V \times A\}$, where $\{V \times A\}$ indicates all combinations of one visual feature with one audio feature.

The multimodal strategy significantly increases the number of candidates due to the multiplicative factor of $|V| \cdot |A|$. While the early experiments in this section of the thesis evaluate the performance of different feature combinations, it would be interesting future work to consider how to efficiently prune the larger candidate set to reduce the cost of feature evaluation. We discuss two possible pruning approaches here.

One naive strategy to handle the additional combinations would be to prune candidates from the combined features whenever either component is pruned as an individual feature, i.e., prune (v_i, a_j) from the candidates whenever v_i is pruned from V or a_j is pruned from A. However, if it takes many iterations to eliminate individual features, the ALM would be performing expensive feature evaluation over the large number of feature combinations for all of these iterations. This leaves the ALM with fewer resources to put towards other important tasks, such as feature extraction and performing inference to generate predictions, as described in Section 5.3.



(a) Early fusion (concatenation) (b) Late fusion

Figure 6.3: Early vs. late fusion of multimodal inputs showing how features from video, audio, and possibly other modalities could be combined.

A second naive strategy would be to only consider a single combination of features consisting of the audio and visual features that perform best individually, i.e., $(v^*, a^*) \in \{V \times A\}$, where $v^* = \arg\max_{v \in V} M(v)$, and $a^* = \arg\max_{a \in A} M(a)$. The ALM would perform feature selection over $V \cup A$, and only when it converges to v^* and a^* would it compare model performance between v^*, a^*, and (v^*, a^*). This approach significantly reduces the number of feature evaluation tasks because only one pair of combined features is considered, rather than $|V| \cdot |A|$ pairs. However, both proposed approaches would fail to select candidates where feature extractors perform better as part of a combination than they do individually, i.e., they would miss candidates (v', a') where $M(v') < M(v^*)$ and $M(a') < M(a^*)$, but $M(v', a') > M(v^*, a^*)$. It is unclear without further analysis and experimentation whether features actually display this behavior where feature extractors that are suboptimal individually become optimal when combined with a feature extractor from a different input mode.

Model training. Recall from Chapter 5 that VOCALExplore trains linear models using the user's labels to predict activities occurring in the unlabeled videos. If VOCALExplore's ALM were to select multimodal combinations of feature extractors, the Model Manager would need to train multimodal models that integrate the information from each feature source.

Multimodal models generally utilize one of two approaches to integrate features [17, 54]: *early fusion* or *late fusion*. We now describe these techniques along with their implications for how the Model Manager would have to be updated.

In *early fusion*, features are combined before training a model. The simplest form of early fusion is concatenation, which stacks the feature vectors from different modalities (Figure 6.3a). However, more complicated approaches may train an embedding model that takes as input feature vectors from the different modalities and outputs a single feature vector. Early fusion enables models to exploit correlations between different input modalities because all features are provided as input to model training. However, early fusion struggles to handle missing modalities (e.g., some videos may lack audio tracks). Further, the concatenation strategy increases feature dimensionality, which may hurt model performance.

The Model Manager would require minimal updates to utilize early fusion; it would simply concatenate the features from all selected feature extractors before training and inference. The actual training and inference logic would not have to be updated.

Late fusion (Figure 6.3b) first trains a model per-modality. A meta-model then combines the decisions or probabilities output from each modality-specific model to generate a final decision. Compared to early fusion, late fusion can more easily adapt to missing features and does not increase the dimensionality of model inputs. However, the models it trains cannot utilize correlations between features from different input modalities.

To utilize late fusion, the Model Manager could train modality-specific models as it does now, i.e., by training linear models over the feature vectors produced by each feature extractor. Note that the output of these linear models is the same regardless of the input: it is a c-dimensional vector that represents the model's confidence that each of the c possible classes appears in the input. The Model Manager would then need to combine the outputs of these modality-specific models. The simplest approach to aggregate the predictions would be to apply pooling over the model outputs. For example, max-pooling would select the most confident prediction for each class across all modality-specific models. This approach would work well if each class can be identified by a single modality, even if different modalities

perform best for different classes. However, combining predictions via pooling ignores possibly useful correlations across outputs. Therefore, it would also be possible for the ModelManager to train an additional meta-model over all modality-specific outputs to learn these correlations. This meta-model can have any architecture, so the simplest implementation would have the Model Manager train yet another linear model for this purpose

6.3.3 Early multimodal results

To evaluate the usefulness of incorporating multimodal features into VOCALExplore, we updated the system to support audio feature extraction and to implement early fusion via feature concatenation. We now present initial results showing the performance of these multimodal models produced by VOCALExplore. We evaluate on three datasets: DEER and K20 from Section 5.4 (Table 5.2), as well as a new dataset: K400-AUDIO. K400-AUDIO consists of a subset of videos from the Kinetics400 dataset [90]; it only contains videos from the classes matching the pattern "playing {instrument}". These classes were chosen because the audio in the video clips generally matches the class label, as verified by manually viewing a random sample of the videos. K400-AUDIO consists of 10948 training videos and 1049 test videos across 21 classes. The training dataset is skewed with the most frequent class appearing in 849 videos and the least frequent class appearing in 218 videos. The full Kinetics400 training dataset contains additional videos for these instrument classes, however, for these experiments we ignore videos lacking an audio track. The validation set is balanced and contains 49-50 videos per class.

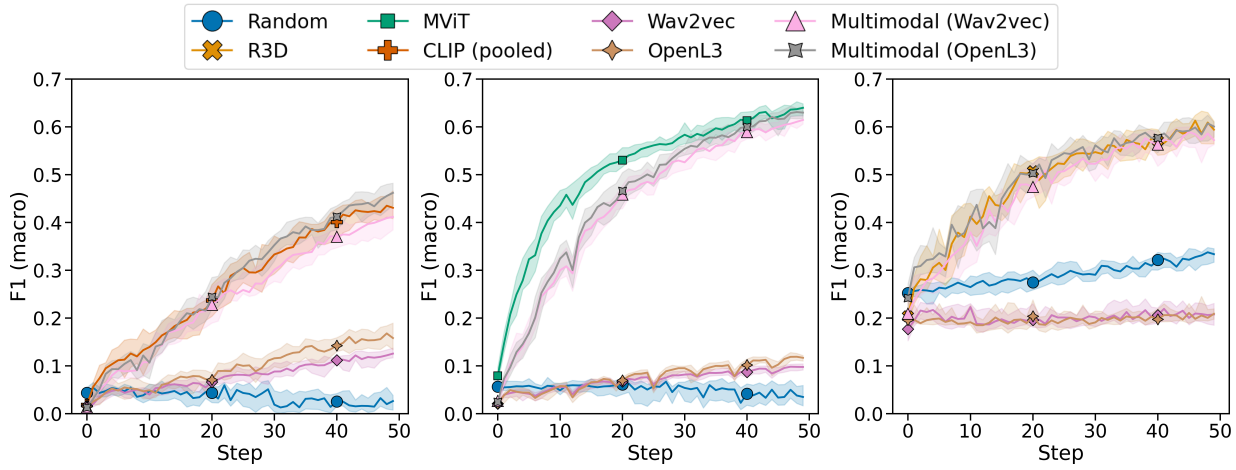
We extract audio features using pretrained Wav2vec 2.0 [19] and OpenL3 [38] models. We use torchaudio [161] and the OpenL3 python package to perform audio feature extraction. The Wav2vec 2.0 model consists of 12 intermediate layers, and we extract the feature representation from the last of these layers which has 768 dimensions. We extract 512-dimensional embeddings from the OpenL3 model trained on "music" content.

Table 6.1: Macro F1 score after training a model on all videos. The table shows the model performance after training only on audio features (wav2ec or OpenL3), only on visual features (R3D for Deer, MViT for K20, and CLIP (Pooled) for K400-Audio), and on a concatenation of the audio and visual features.

Dataset	wav2ec	OpenL3	Visual	Multimodal (wav2ec)	Multimodal (OpenL3)
K400-Audio	0.382	0.577	0.778	0.800	0.817
K20	0.223	0.290	0.795	0.803	0.806
Deer	0.245	0.245	0.799	0.788	0.802

These experiments are intended to explore the usefulness of incorporating audio features into VOCALExplore. They do not attempt to evaluate or address any of the implementation questions highlighted in the previous subsection. In our simplified setup, we evaluate each dataset on the wav2ec and OpenL3 audio features, on the best visual features from Table 5.3, and on the concatenation of each of wav2ec and OpenL3 with the best visual feature. For Deer, the best visual feature extractor is R3D; for K20 it is MViT; and for K400-Audio we use CLIP (Pooled) since MViT and R3D were pretrained on Kinetics400 (which is a superset of K400-Audio). The concatenated features represent the simplest form of early fusion (Figure 6.3a), and are represented in the graphs as "Multimodal (wav2ec)" and "Multimodal (OpenL3)".

Maximum feature extractor performance. We first evaluate the performance of the different feature combinations when training models on the entire training set to understand the maximum expected performance. Table 6.1 shows the macro F1 score for the feature combinations described above. First, looking at the K400-Audio dataset, we observe the model performance of both wav2ec and OpenL3 is well below that of the best visual feature, CLIP (Pooled). Features extracted from OpenL3 generate better models than those from wav2ec for K400-Audio; this is expected because OpenL3 was trained over musical content, which closely matches K400-Audio's domain, while wav2ec was trained over read speech. Despite OpenL3 achieving reasonable performance on its own for K400-Audio,



(a) K400-Audio (b) K20 (c) Deer

Figure 6.4: Multimodal model performance across Explore steps using $B = 5$ and $t = 1$. The line shows the median macro F1 score, and the shaded region shows the interquartile range.

concatenating it with CLIP (Pooled) only modestly improves performance over CLIP (Pooled) on its own. Further, Multimodal (OpenL3) performs only slightly better than Multimodal (wav2vec) despite OpenL3 performing significantly better than wav2vec individually. This indicates that the audio features are not providing additional information to the model beyond what the visual features convey. It is possible that implementing more sophisticated methods of early fusion would improve multimodal performance.

Now looking at K20 and Deer, we observe that neither wav2vec nor OpenL3 produces high-quality models on its own. We see the same behavior as in K400-Audio where models trained on the concatenated multimodal features perform similarly to models trained solely on the best visual feature. This indicates that, given sufficient training data, the models that VOCALExplore trains learn to focus on the informative subset of features.

Iterative feature extractor performance. We now evaluate the quality of models trained using each feature set over a sequence of labeling iterations. We simulate a sequence of 50 Explore steps with $B = 5$ and $t = 1$, which represents 5 1-second video segments being

labeled at each iteration. We use the VE-SAMPLE (CM) method from Section 5.4.2 to determine which video clips to select at each iteration. After each step, we train a model over all labels collected so far and evaluate it on a held out validation set. In this experiment we include the random visual features (RANDOM from Table 5.3) as a baseline for the worst-performing visual feature.

Figure 6.4 shows the results. Looking at K400-AUDIO (Figure 6.4a), we observe that WAV2EC and OPENL3 perform similarly in the few-label regime despite OPENL3 performing significantly better when trained over the entire training set (Table 6.1). We also see that it takes around 30 labeling iterations before the audio features begin to outperform RANDOM. Together, these findings suggest that the audio features are less informative than the visual features and therefore require a larger labeled training set to produce a high-quality model. Figure 6.4a also shows the multimodal features performing similarly to CLIP (POOLED) on its own, which matches the behavior seen when training over the entire training set.

For both K20 (Figure 6.4b) and DEER (Figure 6.4c), the audio features on their own perform poorly. In K20 the audio features perform similarly to RANDOM, while in DEER the audio features perform worse than RANDOM. We observe these results for DEER because although the RANDOM features are extracted from a model initialized with randomized weights (Table 5.3), the model will produce distinctive feature vectors for frames that look different. In contrast, all of the audio tracks in the DEER dataset are similar, so the WAV2EC and OPENL3 feature vectors are also similar and therefore not useful for discriminating classes.

We observe that concatenating the audio and visual features does not improve performance beyond the best visual feature on its own, but it also does not hurt performance as long as there are sufficient labels. For K20, there is a performance gap between MVIT and the multimodal features at early iterations. We likely observe this performance drop at early steps for K20 but not for DEER or K400-AUDIO because the audio in K20 *does* differ across videos, however it generally *does not* correlate with the video contents. Additional training data is thus required for the model to learn which of the features to focus on.

6.3.4 Discussion of multimodal results

The experimental results in the previous section do not indicate that incorporating audio features into VOCALExplore will be immediately beneficial. On two of the datasets (K20 and Deer), adding audio features did not improve model performance over using the best visual feature on its own. Even on K400-Audio, a dataset designed to have informative audio tracks, adding audio features improved performance only minimally over the best visual feature, and it required training over the entire labeled dataset to see this improvement.

The main challenges to improving performance via incorporating multimodal features appear to be twofold. First, many datasets do not have informative audio tracks. This makes it impossible to improve model performance, even given a high-quality pretrained audio model to use as a feature extractor. In these datasets, adding audio features at best cannot improve the model, and at worst will harm model performance by increasing the dimensionality of the features. Second, features from easily accessible pretrained audio models [161, 38] do not produce high-quality models for audio from different domains. For example, wav2ec was pretrained over a speech dataset, and the experimental results show that these features do not transfer well to other domains like identifying musical instruments. This is in contrast to the image and video models used by VOCALExplore that are pretrained on YouTube videos or internet images, and whose features transfer successfully, even to videos as unique as those from the Deer dataset.

It is possible that more sophisticated methods of multimodal fusion beyond feature concatenation could further improve model performance. However, the low accuracy of wav2ec and OpenL3 on their own limits expected improvements because it indicates that there is limited signal in the audio features.

While audio features did not improve performance on the datasets we evaluated on, it is possible that other sources of multimodal features could be helpful. For example, the Deer dataset is accompanied by accelerometer data that captures information about the deer's movement. It is reasonable to hypothesize that this accelerometer data should be

helpful in identifying the deer's activities (e.g., when the deer is bedded, there should be no movement, but when the deer is traveling, there should be maximum movement). However, one bottleneck to utilizing these features in VOCALExplore is the lack of pretrained models over accelerometer data. Currently VOCALExplore's Feature Manager is implemented to extract features from intermediate layers of pretrained models. If such a model does not exist for an input modality, VOCALExplore would need to define a custom feature extractor, which is difficult to do without domain knowledge [57, 26].

Chapter 7

CONCLUSIONS & FUTURE DIRECTIONS

Video data is increasingly easy to capture and store at large scales, leading users from a wide variety of domains to utilize videos for diverse applications. However, current video database management systems (VDBMSs) focus on supporting a subset of common applications that can be implemented by extracting semantic information with off-the-shelf pretrained models. Further, while current VDBMSs recognize that applying these pretrained models to videos is a significant source of query latency, they focus on a limited set of query optimization techniques that fail to exploit common spatial patterns in query workloads. The latter problem leads to inefficient VDBMS storage managers that perform unnecessary preprocessing which increases query latency, while the former problem leaves a large population of potential VDBMS users unserved; these users are unable to benefit from the data management and query processing capabilities provided by current VDBMSs due to the lack of a pretrained model relevant to their videos.

In this thesis we presented two systems to address these challenges. First, in Chapter 4 we introduced TASM, a storage manager for VDBMSs that accelerates queries that process spatial subregions of frames. TASM uses the video codec feature of *tiles* to split frames into independently-decodable regions. This enables TASM to selectively decode and preprocess only those tiles that contain the regions requested by queries, whereas current video storage managers must decode and preprocess entire frames due to spatial dependencies introduced during the video encoding process. TASM is designed to incrementally adapt the tile layout for groups of frames independently as it learns where in frames objects are located and what regions are requested by queries in the workload. Our evaluation of TASM shows

that it can accelerate queries for subregions of frames by an average of 51% and up to 94% while maintaining video quality. TASM appeared in ICDE'21 [41] and is available at `https://github.com/uwdb/TASM`.

Second, in Chapter 5 we introduced VOCALExplore, which is an interactive, pay-as-you-go system to assist users in early video exploration and domain-specific model building. VOCALExplore provides an API with functions to support both manual and system-driven data exploration, and it does not require machine learning expertise. VOCALExplore assists users in building high-quality domain-specific models by choosing how to sample the videos that should be labeled. Further, VOCALExplore automatically selects a feature extractor that performs well on each dataset. Finally, the system implements optimizations to ensure interactive response times to API calls, despite not requiring an expensive preprocessing phase. Our evaluation showed that VOCALExplore is able to automatically match the model performance of the optimal sample selection strategy and feature extractor, all with a user-visible of less than 1 second per API call. VOCALExplore was first proposed in CIDR'22 [42] and will appear at VLDB'24 [43]. Its code is available at `https://github.com/uwdb/VOCALExplore`.

We then presented VOCALExplore's interface in Chapter 6. This interface builds on VOCALExplore's API and reduces friction for common operations like labeling video segments, viewing predictions, and searching over predictions and labels. The interface is an essential component of VOCALExplore that makes it possible for domain experts to interact with the system without having to worry about the underlying API calls. We also discussed possible extensions to VOCALExplore to support additional labeling tasks and give the user more control over label management (Section 6.2) as well as how to incorporate multimodal features (Section 6.3).

TASM and VOCALExplore take steps towards addressing limitations in current VDBMSs that limit their efficiency and usability. However, additional work is required to realize VDBMSs that can support users through the entire lifecycle of a dataset, from ingestion and exploration to complex query processing, all with low query latency and without expensive preprocessing phases to ensure users can quickly generate the insights they care about.

One important open question is how to connect systems such as VOCALExplore that support users in video data ingestion and exploration with systems that support complex query processing over videos [35, 109, 166]. Complex query processing systems enable queries that specify relationships between objects and across time, however they rely on machine learning models to extract semantic information from videos. It remains to be seen whether the domain-specific models produced by VOCALExplore can be used directly by complex query processing systems, or if additional support is needed to bridge the gap. VOCALExplore's models are not optimized for recognizing rare classes or fine-grained categories, so it is possible that refinement steps will be required to produce the semantic insights necessary to execute complex queries.

Another important direction for future research is incorporating multimodal features into VDBMSs to reduce the latency and/or increase the accuracy of query processing. As evaluated in Section 6.3, naive approaches to incorporate audio features into domain-specific models do not improve performance over using just visual features. However, it is important that VDBMSs be able to utilize *all* sources of information that accompany videos beyond just the pixel data, be that audio, metadata (e.g., datetime or location), or accelerometer data. VDBMSs must be able to identify when this auxiliary information is useful and then be able to successfully incorporate it into query processing. There is a rich body of prior work investigating multimodal fusion and model training [17, 54, 157], however the video database community has yet to incorporate and build on this work.

Final remarks: This thesis introduces systems and approaches that improve the storage and analysis of videos. We show that by adapting data management principles to video database management systems we are able to increase the efficiency and usability of VDBMSs. This is an important area of study given the ever increasing number of users with large amounts of video data to analyze.

BIBLIOGRAPHY

[1] Advanced video coding for generic audiovisual services. Rec. ITU-T H.264 and ISO/IEC 14496-10, 06 2019.

[2] 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.

[3] High efficiency video coding. Rec. ISO/IEC 23008-2, Nov 2019.

[4] Xiph.org video test media. media.xiph.org/video/derf, 2019.

[5] Sqlite. https://sqlite.org/index.html, 2020.

[6] Amazon rekognition. https://aws.amazon.com/rekognition/, 2022.

[7] Azure video indexer. https://learn.microsoft.com/en-us/azure/azure-video-indexer/video-indexer-overview, 2022.

[8] Forager: Rapid data exploration and model development. https://cs.stanford.edu/~fpoms/, 2022.

[9] Google cloud video intelligence api. https://cloud.google.com/video-intelligence, 2022.

[10] Nvidia dali. https://developer.nvidia.com/dali, 2023.

[11] PostgreSQL. https://www.postgresql.org, 2023.

[12] wavesurfer.js. https://wavesurfer-js.org, 2023.

[13] Daniel Abadi, Peter A. Boncz, Stavros Harizopoulos, Stratos Idreos, and Samuel Madden. The design and implementation of modern column-oriented database systems. *Found. Trends Databases*, 5(3):197–280, 2013.

[14] Mohammed AbuTaha, Naty Sidaty, Wassim Hamidouche, Olivier Déforges, Jarno Vanne, and Marko Viitanen. End-to-end real-time ROI-based encryption in HEVC videos. In *EUSIPCO*, pages 171–175. IEEE, 2018.

[15] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. Real-time video analytics: The killer app for edge computing. *IEEE Computer*, 50(10):58–67, 2017.

[16] Michael R. Anderson, Michael J. Cafarella, Germán Ros, and Thomas F. Wenisch. Physical representation-based predicate optimization for a visual analytics database. In *IEEE*, pages 1466–1477. IEEE, 2019.

[17] Pradeep K. Atrey, M. Anwar Hossain, Abdulmotaleb El-Saddik, and Mohan S. Kankanhalli. Multimodal fusion for multimedia analysis: a survey. *Multim. Syst.*, 16(6):345–379, 2010.

[18] Pradeep K. Atrey, Mohan S. Kankanhalli, and B. John Oommen. Goal-oriented optimal subset selection of correlated multimedia streams. *ACM Trans. Multim. Comput. Commun. Appl.*, 3(1):2, 2007.

[19] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. In *NeurIPS*, 2020.

[20] Favyen Bastani, Songtao He, Arjun Balasingam, Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan, Michael J. Cafarella, Tim Kraska, and Sam Madden. MIRIS: fast object track queries in video. In *SIGMOD Conference*, pages 1907–1921. ACM, 2020.

[21] Favyen Bastani and Samuel Madden. OTIF: efficient tracker pre-processing over large video datasets. In *SIGMOD*, pages 2091–2104. ACM, 2022.

[22] Sara Beery, Guanhang Wu, Vivek Rathod, Ronny Votel, and Jonathan Huang. Context R-CNN: long term temporal context for per-camera object detection. In *CVPR*, pages 13072–13082. Computer Vision Foundation / IEEE, 2020.

[23] Fabrice Bellard. FFmpeg. `ffmpeg.org`, 2018.

[24] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012.

[25] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[26] Danielle D Brown, Roland Kays, Martin Wikelski, Rory Wilson, and A Peter Klimley. Observing the unwatchable through acceleration logging of animal behavior. *Animal Biotelemetry*, 1:1–16, 2013.

[27] Nicolas Bruno and Surajit Chaudhuri. An online approach to physical design tuning. In *ICDE*, pages 826–835, 2007.

[28] Jiashen Cao, Karan Sarkar, Ramyad Hadidi, Joy Arulraj, and Hyesoon Kim. Figo: Fine-grained query optimization in video analytics. In Zachary Ives, Angela Bonifati, and Amr El Abbadi, editors, *SIGMOD*, pages 559–572. ACM, 2022.

[29] Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. SCOPE: easy and efficient parallel processing of massive data sets. *PVLDB*, 1(2):1265–1276, 2008.

[30] Shih-Fu Chang, William Chen, Horace J. Meng, Hari Sundaram, and Di Zhong. Videoq: An automated content based video search system using visual cues. In *MMSys*, pages 313–324, 1997.

[31] Surajit Chaudhuri and Vivek R. Narasayya. Autoadmin 'what-if' index analysis utility. In *SIGMOD*, pages 367–378, 1998.

[32] Yueting Chen, Nick Koudas, Xiaohui Yu, and Ziqiang Yu. Spatial and temporal constrained ranked retrieval over videos. *PVLDB*, 15(11):3226–3239, 2022.

[33] Yueting Chen, Xiaohui Yu, and Nick Koudas. Ranked window query retrieval over video repositories. In *ICDE*, pages 2776–2791. IEEE, 2022.

[34] Yueting Chen, Xiaohui Yu, Nick Koudas, and Ziqiang Yu. Evaluating temporal queries over video feeds. In *SIGMOD*, pages 287–299. ACM, 2021.

[35] Pramod Chunduri, Jaeho Bang, Yao Lu, and Joy Arulraj. Zeus: Efficiently localizing actions in videos using reinforcement learning. In *SIGMOD*, pages 545–558. ACM, 2022.

[36] Gui Citovsky, Giulia DeSalvo, Claudio Gentile, Lazaros Karydas, Anand Rajagopalan, Afshin Rostamizadeh, and Sanjiv Kumar. Batch active learning at scale. In *NeurIPS*, pages 11933–11944, 2021.

[37] Douglas Comer. The ubiquitous b-tree. *ACM Comput. Surv.*, 11(2):121–137, 1979.

[38] Jason Cramer, Ho-Hsiang Wu, Justin Salamon, and Juan Pablo Bello. Look, listen, and learn more: Design choices for deep audio embeddings. In *ICASSP*, pages 3852–3856. IEEE, 2019.

[39] Isabel F. Cruz and Wendy T. Lucas. Delaunay$^{\mathrm{mm}}$: A visual framework for multimedia presentation. In *VL*, pages 216–223, 1997.

[40] Debabrata Dash, Verena Kantere, and Anastasia Ailamaki. An economic model for self-tuned cloud caching. In *ICDE*, pages 1687–1693, 2009.

[41] Maureen Daum, Brandon Haynes, Dong He, Amrita Mazumdar, and Magdalena Balazinska. TASM: A tile-based storage manager for video analytics. In *ICDE*, pages 1775–1786. IEEE, 2021.

[42] Maureen Daum, Enhao Zhang, Dong He, Magdalena Balazinska, Brandon Haynes, Ranjay Krishna, Apryle Craig, and Aaron Wirsing. VOCAL: video organization and interactive compositional analytics. In *CIDR*, 2022.

[43] Maureen Daum, Enhao Zhang, Dong He, Stephen Mussmann, Brandon Haynes, Ranjay Krishna, and Magdalena Balazinska. Vocalexplore: Pay-as-you-go video data exploration and model building [technical report], 2023.

[44] Peter de Rivaz and Jack Haughton. AV1 bitstream & decoding process specification. *The Alliance for Open Media*, page 182, 2018.

[45] Justin Dellinger, Carolyn Shores, Apryle Craig, Shannon Kachel, Michael Heithaus, William Ripple, and Aaron Wirsing. Predators reduce niche overlap between sympatric prey. *Oikos*, 12 2021.

[46] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*. IEEE Computer Society, 2009.

[47] Kyriaki Dimitriadou, Olga Papaemmanouil, and Yanlei Diao. AIDE: an active learning-based approach for interactive data exploration. *IEEE TKDE*, 28(11):2842–2856, 2016.

[48] Mehmet Emin Dönderler, Ediz Saykol, Umut Arslan, Özgür Ulusoy, and Ugur Güdükbay. Bilvideo: Design and implementation of a video database management system. *MTAP*, 27(1):79–104, 2005.

[49] Haoqi Fan, Tullie Murrell, Heng Wang, Kalyan Vasudev Alwala, Yanghao Li, Yilei Li, Bo Xiong, Nikhila Ravi, Meng Li, Haichuan Yang, Jitendra Malik, Ross Girshick, Matt Feiszli, Aaron Adcock, Wan-Yen Lo, and Christoph Feichtenhofer. PyTorchVideo: A deep learning library for video understanding. In *Proceedings of the 29th ACM International Conference on Multimedia*, 2021. `https://pytorchvideo.org/`.

[50] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. In *ICCV*, pages 6804–6815. IEEE, 2021.

[51] Li Fei-Fei and Ranjay Krishna. Searching for computer vision north stars. *Daedalus*, 151(2):85–99, 2022.

[52] Christoph Feichtenhofer, Haoqi Fan, Bo Xiong, Ross B. Girshick, and Kaiming He. A large-scale study on unsupervised spatiotemporal representation learning. In *CVPR*, pages 3299–3309. Computer Vision Foundation / IEEE, 2021.

[53] Daniel Y. Fu, Will Crichton, James Hong, Xinwei Yao, Haotian Zhang, Anh Truong, Avanika Narayan, Maneesh Agrawala, Christopher Ré, and Kayvon Fatahalian. Rekall: Specifying video events using compositions of spatiotemporal labels. *CoRR*, abs/1910.02993, 2019.

[54] Konrad Gadzicki, Razieh Khamsehashari, and Christoph Zetzsche. Early vs late fusion in multimodal convolutional neural networks. In *FUSION*, pages 1–6. IEEE, 2020.

[55] Ralph Gasser, Luca Rossetto, Silvan Heller, and Heiko Schuldt. Cottontail DB: an open source database system for multimedia retrieval and analysis. In *MM*, pages 4465–4468. ACM, 2020.

[56] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, pages 3354–3361, 2012.

[57] Vincenzo Gervasi, Sven Brunberg, and Jon E Swenson. An individual-based method to measure animal activity levels: A test on brown bears. *Wildlife Society Bulletin*, 34(5):1314–1319, 2006.

[58] Ivan Giangreco and Heiko Schuldt. ADAM pro : Database support for big multimedia retrieval. *Datenbank-Spektrum*, 16(1):17–26, 2016.

[59] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, pages 580–587. IEEE Computer Society, 2014.

[60] Cathal Gurrin, Liting Zhou, Graham Healy, Björn Þór Jónsson, Duc-Tien Dang-Nguyen, Jakub Lokoc, Minh-Triet Tran, Wolfgang Hürst, Luca Rossetto, and Klaus Schöffmann. Introduction to the fifth annual lifelog search challenge, lsc'22. In *ICMR*, pages 685–687. ACM, 2022.

[61] Felix Halim, Stratos Idreos, Panagiotis Karras, and Roland H. C. Yap. Stochastic database cracking: Towards robust adaptive indexing in main-memory column-stores. *PVLDB*, 5(6), 2012.

[62] Patrick Hammer, Tony Lofthouse, Enzo Fenoglio, Hugo Latapie, and Pei Wang. A reasoning based model for anomaly detection in the smart city domain. In *IntelliSys*, volume 1251 of *AISC*, pages 144–159, 2020.

[63] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 2020.

[64] Brandon Haynes, Maureen Daum, Dong He, Amrita Mazumdar, Magdalena Balazinska, Alvin Cheung, and Luis Ceze. VSS: A storage system for video analytics. In *SIGMOD*, pages 685–696. ACM, 2021.

[65] Brandon Haynes, Amrita Mazumdar, Armin Alaghi, Magdalena Balazinska, Luis Ceze, and Alvin Cheung. LightDB: A DBMS for virtual reality video. *PVLDB*, 11(10):1192–1205, 2018.

[66] Brandon Haynes, Amrita Mazumdar, Magdalena Balazinska, Luis Ceze, and Alvin Cheung. Visual road: A video data management benchmark. In *SIGMOD*, pages 972–987. ACM, 2019.

[67] Wenjia He and Michael J. Cafarella. Controlled intentional degradation in analytical video systems. In *SIGMOD*, pages 2105–2119. ACM, 2022.

[68] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowl. Based Syst.*, 212:106622, 2021.

[69] M Heithaus, L Dill, G Marshall, and B Buhleier. Habitat use and foraging behavior of tiger sharks (galeocerdo cuvier) in a seagrass ecosystem. *Marine Biology*, 140(2):237–248, 2002.

[70] Silvan Heller, Mahnaz Parian, Maurizio Pasquinelli, and Heiko Schuldt. Vitrivr-explore: Guided multimedia collection exploration for ad-hoc video search. In *SISAP*, volume 12440 of *Lecture Notes in Computer Science*, pages 379–386. Springer, 2020.

[71] Silvan Heller, Mahnaz Parian, Maurizio Pasquinelli, and Heiko Schuldt. Vitrivr-explore: Guided multimedia collection exploration for ad-hoc video search. In *SISAP*, volume 12440 of *Lecture Notes in Computer Science*, pages 379–386. Springer, 2020.

[72] Silvan Heller, Loris Sauter, Heiko Schuldt, and Luca Rossetto. Multi-stage queries and temporal scoring in vitrivr. In *ICME*, pages 1–5. IEEE, 2020.

[73] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodík, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B. Gibbons, and Onur Mutlu. Focus: Querying large video datasets with low latency and low cost. In *OSDI*, pages 269–286, 2018.

[74] Bo Hu, Peizhen Guo, and Wenjun Hu. Video-zilla: An indexing layer for large-scale video analytics. In *SIGMOD*, pages 1905–1919. ACM, 2022.

[75] Qi Huang, Petchean Ang, Peter Knowles, Tomasz Nykiel, Iaroslav Tverdokhlib, Amit Yajurvedi, Paul Dapolito IV, Xifan Yan, Maxim Bykov, Chuen Liang, Mohit Talwar, Abhishek Mathur, Sachin Kulkarni, Matthew Burke, and Wyatt Lloyd. SVE: distributed video processing at facebook scale. In *SOSP*, pages 87–103. ACM, 2017.

[76] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodík, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. VideoEdge: Processing camera streams using hierarchical clusters. In *SEC*, pages 115–131, 2018.

[77] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. Database cracking. In *CIDR*, pages 68–78, 2007.

[78] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613. ACM, 1998.

[79] Intel. Intel integrated performance primitives. `software.intel.com/content/www/us/en/develop/documentation/ipp-dev-reference`.

[80] Kevin G. Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In Arthur Gretton and Christian C. Robert, editors, *AISTATS*, volume 51 of *JMLR*, pages 240–248, 2016.

[81] Daniel Kang, Peter Bailis, and Matei Zaharia. Blazeit: Optimizing declarative aggregation and limit queries for neural network-based video analytics. *PVLDB*, 13(4):533–546, 2019.

[82] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: Optimizing deep cnn-based queries over video streams at scale. *Proc. VLDB Endow.*, 10(11):1586–1597, 2017.

[83] Daniel Kang, Edward Gan, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. Approximate selection with guarantees using proxies. *Proc. VLDB Endow.*, 13(11):1990–2003, 2020.

[84] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, Yi Sun, and Matei Zaharia. Accelerating approximate aggregation queries with expensive predicates. *Proc. VLDB Endow.*, 14(11):2341–2354, 2021.

[85] Daniel Kang, John Guibas, Peter D. Bailis, Tatsunori Hashimoto, and Matei Zaharia. TASTI: semantic indexes for machine learning-based queries over unstructured data. In *SIGMOD*, pages 1934–1947. ACM, 2022.

[86] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia. Jointly optimizing preprocessing and inference for dnn-based visual analytics. *PVLDB*, 14(2):87–100, 2020.

[87] Siddharth Karamcheti, Ranjay Krishna, Li Fei-Fei, and Christopher D. Manning. Mind your outliers! investigating the negative impact of outliers on active learning for visual question answering. In *ACL/IJCNLP*, pages 7265–7281. Association for Computational Linguistics, 2021.

[88] Ioannis Katsavounidis. Netflix – "El Fuente" video sequence details and scenes. `cdvl.org/documents/ElFuente_summary.pdf`. Accessed: 2020-06.

[89] Ambika Kaul, Saket Maheshwary, and Vikram Pudi. Autolearn - automated feature generation and selection. In *ICDM*, pages 217–226. IEEE Computer Society, 2017.

[90] Will Kay, João Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset. *CoRR*, abs/1705.06950, 2017.

[91] Teuvo Kohonen. The self-organizing map. *Neurocomputing*, 21(1-3):1–6, 1998.

[92] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in WEKA. *JMLR*, 18:25:1–25:5, 2017.

[93] Nick Koudas, Raymond Li, and Ioannis Xarchakos. Video monitoring queries. In *ICDE*, pages 1285–1296. IEEE, 2020.

[94] Miroslav Kratochvíl, Patrik Veselý, Frantisek Mejzlík, and Jakub Lokoc. Som-hunter: Video browsing with relevance-to-som feedback loop. In *MMM*, volume 11962 of *Lecture Notes in Computer Science*, pages 790–795. Springer, 2020.

[95] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *SIGIR*, pages 3–12. ACM/Springer, 1994.

[96] Yang Li, Jiawei Jiang, Jinyang Gao, Yingxia Shao, Ce Zhang, and Bin Cui. Efficient automatic CASH via rising bandits. In *AAAI*, pages 4763–4771. AAAI Press, 2020.

[97] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. Toward a practical perceptual video quality metric. `https://netflixtechblog.com/653f208b9652`, 06 2016. Accessed: 2020-06-01.

[98] Zhicheng Liu and Jeffrey Heer. The effects of interactive latency on exploratory visual analysis. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2122–2131, 2014.

[99] Cornell Feeders Live, Battling Birds, and Panama Edition. Cornell feeders live final report.

[100] Jakub Lokoc, Patrik Veselý, Frantisek Mejzlík, Gregor Kovalcík, Tomás Soucek, Luca Rossetto, Klaus Schoeffmann, Werner Bailer, Cathal Gurrin, Loris Sauter, Jaeyub Song, Stefanos Vrochidis, Jiaxin Wu, and Björn Þór Jónsson. Is the reign of interactive search eternal? findings from the video browser showdown 2020. *ACM Trans. Multim. Comput. Commun. Appl.*, 17(3):91:1–91:26, 2021.

[101] Andrea Lottarini, Alex Ramírez, Joel Coburn, Martha A. Kim, Parthasarathy Ranganathan, Daniel Stodolsky, and Mark Wachsler. vbench: Benchmarking video transcoding in the cloud. In *ASPLOS*, pages 797–809, 2018.

[102] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vis.*, 60(2):91–110, 2004.

[103] Yao Lu, Aakanksha Chowdhery, and Srikanth Kandula. Optasia: A relational platform for efficient large-scale video analytics. In *SoCC*, pages 57–70. ACM, 2016.

[104] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. Accelerating machine learning inference with probabilistic predicates. In *SIGMOD*, pages 1493–1508. ACM, 2018.

[105] Rachael Mady, Peter Mason, Matt Strimas-Mackey, Miyoko Chu, Tina Phillips, David Bonter, Charles Eldermire, and Benjamin Walters. Bird cams lab biological data. `https://ecommons.cornell.edu/handle/1813/110264`, 2021.

[106] TorchVision maintainers and contributors. TorchVision: PyTorch's Computer Vision library, 11 2016.

[107] Anqi Mao, Mehryar Mohri, and Yutao Zhong. Cross-entropy loss functions: Theoretical analysis and applications. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 23803–23828. PMLR, 2023.

[108] Amrita Mazumdar, Brandon Haynes, Magda Balazinska, Luis Ceze, Alvin Cheung, and Mark Oskin. Perceptual compression for video storage and processing systems. In *SoCC*, pages 179–192, 2019.

[109] Stephen Mell, Favyen Bastani, Stephan Zdancewic, and Osbert Bastani. Synthesizing video trajectory queries. In *AIPLANS Workshop*, 2021.

[110] Anton Milan, Laura Leal-Taixé, Ian D. Reid, Stefan Roth, and Konrad Schindler. MOT16: A benchmark for multi-object tracking. *CoRR*, abs/1603.00831, 2016.

[111] Kiran M. Misra, C. Andrew Segall, Michael Horowitz, Shilin Xu, Arild Fuldseth, and Minhua Zhou. An overview of tiles in HEVC. *Journal of Selected Topics in Signal Processing*, 7(6):969–977, 2013.

[112] Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. Analyzing and mitigating data stalls in DNN training. *PVLDB*, 14(5):771–784, 2021.

[113] Oscar R. Moll, Favyen Bastani, Sam Madden, Mike Stonebraker, Vijay Gadepally, and Tim Kraska. Exsample: Efficient searches on video repositories through adaptive sampling. In *ICDE*, pages 2956–2968. IEEE, 2022.

[114] Ravi Teja Mullapudi, Fait Poms, William R. Mark, Deva Ramanan, and Kayvon Fatahalian. Learning rare category classifiers on a tight labeling budget. In *ICCV*, pages 8403–8412. IEEE, 2021.

[115] Thomas Neumann. Efficiently compiling efficient query plans for modern hardware. *Proc. VLDB Endow.*, 4(9):539–550, 2011.

[116] Omar Aziz Niamut, Emmanuel Thomas, Lucia D'Acunto, Cyril Concolato, Franck Denoual, and Seong Yong Lim. MPEG DASH SRD: spatial relationship description. In *MMSys*, pages 5:1–5:8, 2016.

[117] Nvidia video codec. `developer.nvidia.com/nvidia-video-codec-sdk`.

[118] Eitetsu Oomoto and Katsumi Tanaka. OVID: design and implementation of a video-object database system. *TKDE*, 5(4):629–643, 1993.

[119] Albert Orriols-Puig and Ester Bernadó-Mansilla. Evolutionary rule-based systems for imbalanced data sets. *Soft Comput.*, 13(3):213–225, 2009.

[120] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[121] Alex Poms, Will Crichton, Pat Hanrahan, and Kayvon Fatahalian. Scanner: efficient video analysis at scale. *ACM Trans. Graph.*, 37(4):138, 2018.

[122] Mark Raasveldt and Hannes Muehleisen. DuckDB.

[123] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang, editors, *ICML*, volume 139, pages 8748–8763. PMLR, 2021.

[124] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[125] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(6):1137–1149, 2017.

[126] Francisco Romero, Johann Hauswald, Aditi Partap, Daniel Kang, Matei Zaharia, and Christos Kozyrakis. Optimizing video analytics with declarative model relationships. *PVLDB*, 16(3):447–460, 2022.

[127] Luca Rossetto, Ivan Giangreco, and Heiko Schuldt. Cineast: A multi-feature sketch-based video retrieval engine. In *ISM*, pages 18–23. IEEE, 2014.

[128] Luca Rossetto, Ivan Giangreco, Claudiu Tanase, and Heiko Schuldt. vitrivr: A flexible retrieval stack supporting multiple query modes for searching in multimedia collections. In *ACM Multimedia*, pages 1183–1186. ACM, 2016.

[129] Madeline C Schiappa, Yogesh S Rawat, and Mubarak Shah. Self-supervised learning for videos: A survey. *ACM Computing Surveys*.

[130] Fritz W Scholz and Michael A Stephens. K-sample anderson–darling tests. *Journal of the American Statistical Association*, 82(399):918–924, 1987.

[131] Andy Schuler and Matthew Donato. Engineers making movies (AKA open source test content). `netflixtechblog.com/f21363ea3781`, 2018. Accessed: 2020-06.

[132] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access path selection in a relational database management system. In *SIGMOD*, pages 23–34. ACM, 1979.

[133] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *ICLR*, 2018.

[134] Andrew W Senior, L Brown, Arun Hampapur, C-F Shu, Yun Zhai, Rogério Schmidt Feris, Y-L Tian, Sergio Borger, and C Carlson. Video analytics for retail. In *AVSS*, pages 423–428, 2007.

[135] Burr Settles. Active learning literature survey. 2009.

[136] Lifeng Shang, Linjun Yang, Fei Wang, Kwok-Ping Chan, and Xian-Sheng Hua. Real-time large scale near-duplicate web video retrieval. In Alberto Del Bimbo, Shih-Fu Chang, and Arnold W. M. Smeulders, editors, *MM*, pages 531–540. ACM, 2010.

[137] Amit P. Sheth, Clemens Bertram, and Kshitij Shah. Videoanywhere: A system for searching and managing distributed video assets. *SIGMOD Record*, 28(1):104–109, 1999.

[138] Gunnar A. Sigurdsson, Gül Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *ECCV*, volume 9905 of *Lecture Notes in Computer Science*, pages 510–526. Springer, 2016.

[139] Lucas Smaira, João Carreira, Eric Noland, Ellen Clancy, Amy Wu, and Andrew Zisserman. A short note on the kinetics-700-2020 human action dataset. *CoRR*, abs/2010.10864, 2020.

[140] Manuel Stein, Halldor Janetzko, Andreas Lamprecht, Thorsten Breitkreutz, Philipp Zimmermann, Bastian Goldlücke, Tobias Schreck, Gennady L. Andrienko, Michael Grossniklaus, and Daniel A. Keim. Bring it to the pitch: Combining video and movement data to enhance team sport analysis. *TVCG*, 24(1):13–22, 2018.

[141] Gary J. Sullivan, Jens-Rainer Ohm, Woojin Han, and Thomas Wiegand. Overview of the high efficiency video coding (HEVC) standard. *TCSVT*, 22(12):1649–1668, 2012.

[142] Gary J. Sullivan and Thomas Wiegand. Video compression - from concepts to the H.264/AVC standard. *Proc. IEEE*, 93(1):18–31, 2005.

[143] Abhijit Suprem, Joy Arulraj, Calton Pu, and João Eduardo Ferreira. ODIN: automated drift detection and recovery in video analytics. *PVLDB*, 13(11):2453–2465, 2020.

[144] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *ICANN*, volume 11141 of *Lecture Notes in Computer Science*, pages 270–279. Springer, 2018.

[145] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *CVPR*, pages 6450–6459. IEEE, 2018.

[146] Mario Vranjes, Snjezana Rimac-Drlje, and Kresimir Grgic. Locally averaged psnr as a simple objective video quality metric. In *ELMAR*, volume 1, pages 17–20. IEEE, 2008.

[147] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *ICCV*, pages 3551–3558. IEEE, 2013.

[148] Junjue Wang, Ziqiang Feng, Zhuo Chen, Shilpa Anna George, Mihir Bala, Padmanabhan Pillai, Shao-Wen Yang, and Mahadev Satyanarayanan. Bandwidth-efficient live video analytics for drones via edge computing. In *SEC*, pages 159–173, 2018.

[149] Xiaoli Wang, Aakanksha Chowdhery, and Mung Chiang. Skyeyes: adaptive video streaming from uavs. In *HotWireless@MobiCom*, pages 2–6, 2016.

[150] Jonathan Waring, Charlotta Lindvall, and Renato Umeton. Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artif. Intell. Medicine*, 104:101822, 2020.

[151] Waymo. Waymo open dataset FAQ. `waymo.com/open/faq`, 2021.

[152] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *PVLDB*, pages 194–205. Morgan Kaufmann, 1998.

[153] Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *NIPS*, pages 1753–1760. Curran Associates, Inc., 2008.

[154] Yi Wu, Edward Y. Chang, Kevin Chen-Chuan Chang, and John R. Smith. Optimal multimodal fusion for multimedia data analysis. In *MM*, pages 572–579. ACM, 2004.

[155] Yifan Wu, Steven Mark Drucker, Matthai Philipose, and Lenin Ravindranath. Querying videos using DNN generated labels. In *HILDA@SIGMOD*, pages 6:1–6:6. ACM, 2018.

[156] Ioannis Xarchakos and Nick Koudas. SVQ: streaming video queries. In *SIGMOD*, 2019.

[157] Fanyi Xiao, Yong Jae Lee, Kristen Grauman, Jitendra Malik, and Christoph Feichtenhofer. Audiovisual slowfast networks for video recognition. *CoRR*, abs/2001.08740, 2020.

[158] Hu Xu, Gargi Ghosh, Po-Yao Huang, Dmytro Okhonko, Armen Aghajanyan, Florian Metze, Luke Zettlemoyer, and Christoph Feichtenhofer. Videoclip: Contrastive pre-training for zero-shot video-text understanding. In *EMNLP*, pages 6787–6800, 2021.

[159] Tiantu Xu, Luis Materon Botelho, and Felix Xiaozhu Lin. Vstore: A data store for analytics on large videos. In *EuroSys*, pages 16:1–16:17. ACM, 2019.

[160] Zhuangdi Xu, Gaurav Tarlok Kakkar, Joy Arulraj, and Umakishore Ramachandran. EVA: A symbolic approach to accelerating exploratory video analytics with materialized views. In *SIGMOD*, pages 602–616. ACM, 2022.

[161] Yao-Yuan Yang, Moto Hira, Zhaoheng Ni, Anjali Chourdia, Artyom Astafurov, Caroline Chen, Ching-Feng Yeh, Christian Puhrsch, David Pollack, Dmitriy Genzel, Donny Greenberg, Edward Z. Yang, Jason Lian, Jay Mahadeokar, Jeff Hwang, Ji Chen, Peter Goldsborough, Prabhat Roy, Sean Narenthiran, Shinji Watanabe, Soumith Chintala, Vincent Quenneville-Bélair, and Yangyang Shi. Torchaudio: Building blocks for audio and speech processing. *arXiv preprint arXiv:2110.15018*, 2021.

[162] Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. Taking human out of learning applications: A survey on automated machine learning. *arXiv:1810.13306*, 2018.

[163] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *CVPR*, pages 2636–2645, 2020.

[164] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[165] Sagi Zeevi. BackgroundSubtractorCNT. `https://sagi-z.github.io/BackgroundSubtractorCNT`, 2016. Accessed: 2021-01-21.

[166] Enhao Zhang et al. Equi-vocal: Synthesizing queries for compositional video events from limited user interactions. *PVLDB*, 16(12), 2023.

[167] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodík, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. Live video analytics at scale with approximation and delay-tolerance. In *NSDI*, pages 377–392, 2017.

[168] Huayi Zhang, Lei Cao, Samuel Madden, and Elke A. Rundensteiner. LANCET: labeling complex data at scale. *PVLDB*, 14(11):2154–2166, 2021.

[169] Weihang Zhang, Yuma Kinoshita, and Hitoshi Kiya. Image-enhancement-based data augmentation for improving deep learning in image classification problem. In *ICCE-TW*, pages 1–2. IEEE, 2020.

[170] Yuhao Zhang and Arun Kumar. Panorama: A data system for unbounded vocabulary querying over video. *PVLDB*, 13(4):477–491, 2019.

[171] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proc. IEEE*, 109(1):43–76, 2021.

Appendix A

FREQUENCY-BASED HYPOTHESIS TEST

This section presents the formula that we use to compute the p-value for the frequency-based test described in Section 5.2.1.

Given a multi-class data distribution with k classes, define the class frequencies as $p \in \Delta_k = \{p \in \mathbb{R}_+^k : \sum_{i=1}^k p_i = 1\}$ such that $p_i = \Pr[Y = i]$.

For a given multiplicative threshold $m \geq 1$, we say class frequencies p are "imbalanced" if

$$\min_i p_i < \frac{1}{mk} \tag{A.1}$$

Define the imbalanced and balanced distributions as,

$$P_{\text{imbalanced}} = \left\{ p \in \Delta_k : \min_i p_i < \frac{1}{mk} \right\} \tag{A.2}$$

$$P_{\text{balanced}} = \left\{ p \in \Delta_k : \min_i p_i \geq \frac{1}{mk} \right\} \tag{A.3}$$

Denote an empirical count vector as $C \in \mathbb{Z}_+^k$ where $n = \sum_i C_i$ is the total count. We define the test statistic as $\phi(C) = \min_i C_i$. We will say the distribution is imbalanced if $\phi(C)$ is sufficiently small (say less than or equal to a threshold t). We want to bound the False Discovery Rate (FDR) by a probability such as 5% or 1%. For a threshold $t \in \mathbb{Z}_+$, the worst-case FDR is,

$$\max_{p \in P_{\text{balanced}}} \Pr_{C \sim \text{Multinomial}(n,p)}(\phi(C) \leq t) = \tag{A.4}$$

$$= \max_{p \in P_{\text{bal}}} \Pr_{C \sim \text{Multinomial}(n,p)}(\exists i : C_i \leq t) \tag{A.5}$$

$$\leq \max_{p \in P_{\text{balanced}}} \sum_{i=1}^{k} \Pr_{C \sim \text{Multinomial}(n,p)}(C_i \leq t) \tag{A.6}$$

$$\leq \sum_{i=1}^{k} \max_{p \in P_{\text{balanced}}} \Pr_{C \sim \text{Multinomial}(n,p)}(C_i \leq t) \tag{A.7}$$

$$= \sum_{i=1}^{k} \Pr\left(\text{Binomial}\left(n, \frac{1}{mk}\right) \leq t\right) \tag{A.8}$$

$$= k \Pr\left(\text{Binomial}\left(n, \frac{1}{mk}\right) \leq t\right) \tag{A.9}$$

Thus, the "p-value" with n samples and test statistic $\phi(C)$ is bounded by:

$$k \Pr\left(\text{Binomial}\left(n, \frac{1}{mk}\right) \leq \phi(C)\right) \tag{A.10}$$

We implement this in Python as:

```
p_value = k * scipy.stats.binom.cdf( min(C), n, 1/(m*k) )
```