FiGO: Fine-Grained Query Optimization in Video Analytics

Jiashen Cao
Georgia Institute of Technology
jiashenc@gatech.edu

Karan Sarkar
Georgia Institute of Technology
ksarkar9@gatech.edu

Ramyad Hadidi
Georgia Institute of Technology
rhadidi@gatech.edu

Joy Arulraj
Georgia Institute of Technology
arulraj@gatech.edu

Hyesoon Kim
Georgia Institute of Technology
hyesoon@cc.gatech.edu

Abstract

Video database management systems (VDBMSs) enable automated analysis of videos at scale using computationally-intensive deep learning models. To reduce the computational overhead of these models, researchers have proposed two techniques: (1) leveraging a specialized, lightweight model to filter out irrelevant frames or to directly answer the query, and (2) using a cascade of models of increasing complexity to answer the query. For both techniques, the query optimizer generates a coarse-grained query plan for the entire video. These techniques suffer from four limitations: (1) lower query accuracy over hard-to-detect predicates, (2) lower filtering efficacy with frequently-occurring objects, (3) lower accuracy due to non-trivial model cascade configuration, and (4) missed optimization opportunities due to coarse-grained planning for the entire video.

In this paper, we present FiGO to tackle these limitations. The design of FiGO is centered around three techniques. First, it uses an ensemble of models to support a range of throughput-accuracy tradeoffs. Second, it adopts a fine-grained approach to query optimization. It processes different chunks of the video using different models in the given ensemble to meet the user's accuracy requirement. Lastly, it uses a lightweight technique to prune the model ensemble to lower the query optimization time. We empirically show that these techniques enable FiGO to outperform the state-of-the-art systems for processing queries over videos by 3.3× on average across four video datasets.

CCS Concepts

• **Information systems** → *Query optimization*; • **Computing methodologies** → *Object detection.*

Keywords

Video Analytics

ACM Reference Format:
Jiashen Cao, Karan Sarkar, Ramyad Hadidi, Joy Arulraj, and Hyesoon Kim. 2022. FiGO: Fine-Grained Query Optimization in Video Analytics. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3514221.3517857

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA.

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3517857

1 INTRODUCTION

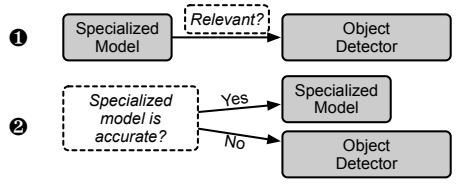
By leveraging recent advances in deep learning (DL), video database management systems (VDBMSs) enable automated analysis of videos at scale. These systems use 2-D object detection models for locating objects of interest in the videos. The runtime performance of these systems is constrained by the computational overhead associated with processing the frames using the DL model. For example, the Det-7 model from the EfficientDet family of object detectors [28] only processes 4 frames per second on Titan XP GPU.

To speed up query execution, researchers have proposed two techniques for reducing the invocation of the heavyweight, object detection model (*a.k.a.,* reference model [1]) in the VDBMS. The first technique consists of using a lightweight, specialized model [13, 14, 19] for quickly filtering out irrelevant frames. The second technique consists of using a sequence of lightweight models [2]. We next describe these two approaches in detail:

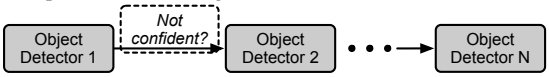
Model Specialization (MS). This technique consists of using a lightweight, specialized model to accelerate query processing. As shown in Figure 1a, a VDBMS may use a specialized model to filter out irrelevant frames or to even directly answer the query. By reducing the number of invocations of the heavyweight, reference model, the VDBMS accelerates the query with a tolerable drop in accuracy. Two exemplars of this technique are PP [19] and BlazeIt [13]. PP filters out irrelevant frames using the specialized model (❶ in Figure 1a). The frames that are considered relevant are subsequently processed using the reference model. We refer to this technique of using specialized models as MS-Filter. In contrast, BlazeIt uses the specialized model to directly answer the given query (❷ in Figure 1a). It only uses the reference model if the specialized model is not accurate enough. We refer to this technique of using specialized models as MS-Skip.

Model Cascade (MC). Another approach for efficiently processing video analytics queries consists of using a sequence of models, called a model cascade, as shown in Figure 1b. While processing the query, the VDBMS short-circuits the inference based on the feedback returned by each model (*e.g.,* confidence score). An exemplar of this technique is the Tahoma system [2].

[1] We refer to the most accurate (and often also the most compute-intensive model) in the model pipeline as the reference model.



(a) Model Specialization – Configurations associated with MS-Filter and MS-Skip.



(b) Model Cascade – Configuration associated with MC.

Figure 1: Techniques for Accelerating Queries – Two approaches taken for accelerating queries in state-of-the-art video analytics systems.

1.1 Limitations

While these techniques successfully accelerate queries, they still suffer from four limitations. We highlight these limitations using the following illustrative query:

```
SELECT frameID FROM UA-DeTrac WHERE Count(Bus)≥1;
```

I – model specialization overhead. In MS-Filter, since each filter detects only one object category (*e.g.,* bus), the VDBMS needs to train several specialized models (*i.e.,* filters) at runtime. In contrast, with BlazeIt, since a specialized model directly returns the number of buses in an image, it must maintain a separate model for each predicate (*e.g.,* `Count(Bus)`). Thus, with model specialization, VDBMSs need to train and maintain a large collection of models for different objects and predicates, respectively. The overhead of training a specialized model while processing a previously unseen query is significant. For instance, training a ResNet-34 model on 100 sampled video frames takes 32 seconds.

II – high selectivity query. The speedup obtained with the filtering technique used in MS-Filter relies on the data reduction rate. Consider a video with N frames. Let the fraction of frames discarded by the specialized model be r, and the costs of running the filter and running the reference model be C_f and C_o per frame, respectively. To accelerate the query, r must satisfy this constraint:

$$N(C_f + (1-r)\cdot C_o) < NC_o$$
$$r > \frac{C_f}{C_o}$$

This constraint is not met by queries with high selectivity. To illustrate this problem, we replicate MS-Filter using a model from the EfficientDet family of object detectors [28]. We defer a description of the empirical setup to §7. The results are shown in Figure 2. MS-Filter takes more time to process queries with high selectivity (*e.g.,* $r = 0.2$). Thus, when the data reduction rate is small, the performance gap between MS-Filter and Naive (*i.e.,* naively running object detector on every frame) is minimal.

We replicate the model cascade (MC) approach using eight models from EfficientDet. This approach also does not work well on queries with high selectivity. This is because a large fraction of video frames *cannot* be filtered out by earlier models in the cascade leading to slower query processing.

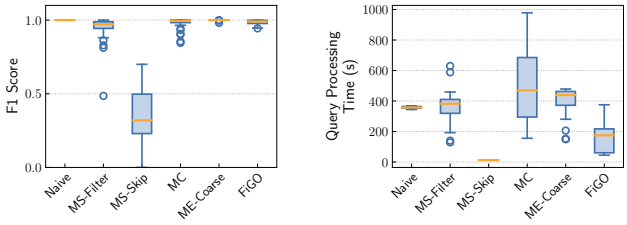


Figure 2: Comparison of Video Analytics Systems – F-1 score and query processing time associated with handling the query in Listing 1.1 across different video analytics systems.

III – difficult-to-detect predicates. MS-Skip [13] uses a specialized model to directly return aggregates (*e.g.,* number of cars in an image). This approach does not generalize to all predicates. First, the specialized model is designed to be shallow for faster execution. So, it is unable to deliver high accuracy for harder predicates (*e.g.,* small objects in the background of the frame). Second, it relies on a subset of the videos for training. Lack of positive examples in the selected subset greatly affects the quality of the model. We replicate MS-Skip by training a ResNet-34 model [9]. As shown in Figure 2, MS-Skip is faster than Naive. But, its F-1 score is worser than its counterparts. While it delivers 0.7 F-1 score on some videos, it fails to provide useful results on others.

We observe that the model cascade approach also suffers from accuracy loss. First, short-circuiting based on confidence-scores does not provide a reliable way to achieve good accuracy. Second, the confidence score thresholds fail to generalize to the entire video.

IV – Coarse-Grained Optimization. Another drawback of state-of-the-art DBMSs is that they adopt a coarse-grained approach towards finding the *model configuration* (*e.g.,* a filter followed by the reference model or a model cascade) to process the query. We refer to the chosen configuration as a *query plan*. For instance, the Optimizer may uniformly sample 10% of the frames from the video and select a fixed configuration for the entire video. Given an accuracy constraint, coarse-grained optimization leads to higher query processing time. Positive events tend to not appear in every segment of the video, especially with low selectivity queries. If the Optimizer picks the same query plan for the entire video, video segments that are less likely to contain positive events or those that contain easy-to-detect events are processed using the same slow model configuration tailored for important segments.

1.2 FiGO

We present FiGO (**Fi**ne-**G**rained query **O**ptimization in video analytics), a VDBMS that addresses the limitations highlighted above using a novel model ensemble approach together with fine-grained optimization. In FiGO, the Optimizer first splits the given video into a sequence of *chunks* of varying sizes. It then picks an appropriate model from a collection of models for each chunk. Unlike the model cascade approach, the Execution Engine in FiGO only processes a chunk with one model picked from the ensemble.

Prior efforts do not focus on query optimization across a model ensemble. This is critical since there are several suitable models for a given vision task. For instance, Faster-RCNN [25] and SSD [18] object detectors offer different performance-accuracy tradeoffs. We

present a novel technique for optimizing queries using a collection of models. Another limitation of the model specialization and cascade approaches is that they require modifications to the neural network's architecture or the model configuration. In contrast, FiGO uses off-the-shelf models to accelerate queries. It does not train any specialized models to process ad-hoc queries. This enables it to seamlessly work across diverse queries and video datasets.

FiGO takes a fine-grained approach to query optimization. Since the content of a video often changes, the optimal model for processing the chunk also varies. FiGO tailors the plan for each chunk. It uses slower, more accurate models for processing important chunks. It either skips or applies faster, less accurate models for processing irrelevant chunks. FiGO uses a novel accuracy-driven sample size bound to determine the sizes of chunks [2]. The Optimizer initially treats the entire video as a single chunk and then iteratively splits it into smaller chunks depending on the accuracy constraint.

We illustrate the benefits of the fine-grained approach by constructing a baseline that couples a model ensemble approach with coarse-grained optimization. In this case, the VDBMS picks a single optimal model from the ensemble to process the entire video. We refer to this approach as ME-Coarse. As shown in Figure 2, to meet the target accuracy constraint, ME-Coarse often picks a compute-intensive model, even though many segments in the video do not need such a model. This leads to a higher query processing time on most of the videos. In contrast, FiGO tailors the model configuration for each chunk in the video, thereby delivering lower query processing time while still meeting the accuracy constraint.

Coupling a model ensemble approach with fine-grained optimization leads to a higher query optimization time. The Optimizer must profile the sampled frames using all the models in the ensemble. To lower this optimization overhead that prior VDBMSs do not suffer from, we introduce a novel technique for pruning out a subset of models with limited utility. We demonstrate that FiGO outperforms state-of-the-art VDBMSs across diverse queries and video datasets with respect to both accuracy *and* performance.

Contributions. FiGO makes the following contributions.

- FiGO adopts a novel model ensemble approach coupled with fine-grained optimization to accelerate query processing.
- FiGO leverages an accuracy-driven bound for chunking the given video that strikes a balance between query optimization time and query execution time.
- FiGO prunes the set of models using a variant of Thompson sampling to reduce optimization time.
- FiGO is 3.3× faster on average compared to the state-of-the-art VDBMSs across diverse queries. It generalizes to four different video datasets by not relying on ad-hoc, specialized models.

2 BACKGROUND

In this section, we provide an overview of the optimizations employed in state-of-the-art VDBMSs [2–4, 8, 10, 12, 12–14, 16, 19, 22, 32, 34]. Table 1 lists the key characteristics of these VDBMSs: (1) PP [19], (2) BlazeIt [13], (3) NoScope [14], (4) Tahoma [2], (5) Panorama [34], and (6) Miris [3].

[2]Like other VDBMSs, we measure accuracy with respect to the most accurate model (*a.k.a.,* reference model) in the ensemble.

System	Execution			Optimization	
	+ MS	+ MC	+ ME †	Coarse	Fine †
PP	✔			✔	
BlazeIt	✔			✔	
NoScope	✔			✔	
Tahoma		✔		✔	
Panorama		✔		✔	
Miris					✔
FiGO			✔		✔

MS: Model Specialization, **MC**: Model Cascade
ME: Model Ensemble, †: Techniques used in FiGO.

Table 1: Qualitative Comparison of Video Analytics Systems – Key characteristics of state-of-the-art VDBMSs.

As we discussed in §1, PP, BlazeIt, and NoScope accelerate queries using model specialization. They construct lightweight models in an ad-hoc manner to answer the query. To choose which specialized model to use, they evaluate the models on a set of sampled frames during query optimization.

Tahoma [2] is a closely related VDBMS. It constructs a model cascade by combining a chain of models (*e.g.,* image classification or object detection models) and determines when to short-circuit the inference based on the confidence score of prediction of each model in the chain. Tahoma speeds up queries by skipping compute-intensive models that appear at the end of the chain. Unlike Tahoma, FiGO does not use a model cascade. Instead, for each chunk, FiGO picks only one model to use.

Panorama [34] is another state-of-the-art VDBMS that uses a single cascaded model to solve the unbounded vocabulary problem in object recognition. Similar to Tahoma, it offers a set of performance-accuracy tradeoffs. The model generates multiple feature embeddings for an input with different levels of quality. Panorama determines the appropriate performance-accuracy tradeoff point based on the delta between the representative embedding of a category and the embedding of the input.

Miris [3] is a VDBMS focused on multi-object tracking. It uses a recurrent neural network and a graph neural network to mark the tracking trajectories between objects. Miris also adopts a fine-grained approach to tuning the tracking accuracy. It starts sampling at a low frame rate to gain a high-level perspective of the video at the beginning and then gradually increases the sampling rate to improve the accuracy of tracking. FiGO differs from Miris in two ways. First, FiGO takes a fine-grained approach towards both query optimization and query processing. Second, it is tailored for object detection instead of tracking.

3 OUR APPROACH

In this section, we first describe how FiGO couples fine-grained optimization with a model ensemble approach in §3.1. We discuss the importance of adaptively changing the optimal model for each video chunk in lowering query processing time and improving accuracy. We conclude with an overview of FiGO in §3.2.

3.1 Fine-Grained Optimization over Ensemble

Chunk. A chunk consists of a contiguous sequence of video frames. We denote a chunk by V_i. Chunks may vary in their size (*i.e.,* number of frames). The maximum size of a chunk is the size of original video V. A video V with R chunks may be specified as:

$$|V| = \sum_{i=1}^{R} |V_i|, \quad 0 < |V_i| \leq |V|$$

Two key design decision in FiGO are: (1) it operates at the chunk-granularity for both optimization and execution. (2) it always picks a *fixed* number of samples from a chunk for query optimization. To obtain more samples in an interesting part of the video, it splits that part into a larger set of fine-grained chunks. Similarly, it maps an uninteresting part of the video to a single, coarse-grained chunk. The Optimizer uses sample size estimation and cost estimation to determine how to split the video into a sequence of chunks (§4).

Query Execution. The Execution Engine uses a model ensemble M with $|M|$ models (*e.g.,* object detectors). Unlike the model specialization approach, there is no online training overhead in FiGO since it does not construct ad-hoc models. Unlike the model cascade approach, the Execution Engine only processes each chunk with exactly one model. FiGO delivers lower query processing time as it only uses the optimal model for processing the chunk. The Execution Engine completely skips processing irrelevant chunks (*i.e.,* does not run any of the models in the ensemble over them). Thus, FiGO accelerates queries by processing less interesting chunks using faster models and skipping irrelevant chunks.

Query Optimization. The Optimizer is responsible for picking the optimal model configuration for each chunk. In FiGO, the query plan consists of exactly one model or skipping the chunk. Unlike other VDBMSs, FiGO splits the video into a set of chunks. The size of each chunk depends on: (1) the estimated number of required samples, and (2) the estimated execution time for the chunk. After splitting, the Optimizer evaluates potential plans for each chunk. Given a chunk, it first picks a few samples $\hat{V}_i$ and then evaluates each model m' from the ensemble M over $\hat{V}_i$ to obtain accuracy and processing time metrics $f(x, m')$:

$$\hat{V}_i \rightarrow \{\, f(x, m') \mid x \in \hat{V}_i, m' \in M \,\}$$

For simplicity, FiGO uses uniform sampling during this *profiling* step. Lastly, the Optimizer constructs the set of optimal plans P for all of the chunks:

$$P \sim \{\, [V_1, m_2], [V_2, m_1] \ldots [V_R, m_3] \,\}$$

Since the Optimizer needs to profile multiple models during optimization, the profiling step increases query optimization time. We will later present a model ensemble pruning technique for lowering this overhead.

Case Study. We illustrate how FiGO processes the query in Figure 3. In this example, we assume the VDBMS only has access to an ensemble with three models (*e.g.,* Det-0, Det-1, and Det-2 from EfficientDet [28]). Det-0 is the fastest model and Det-2 is the most accurate model (*a.k.a.,* the reference model). We use the same query that checks for the existence of a bus object in a frame.

The Optimizer must make two decisions. First, it must determine how to split the video into chunks. Second, it must decide which model to use for each chunk or to skip that chunk. To make these decisions, the Optimizer always picks the same number of video frames from each chunk (denoted by λ, *e.g.,* λ = 10 frames). It then estimates the number of required samples to find the optimal model (elaborated in §4). After confirming that the sample size in a given chunk is sufficient, it picks the model to use based on the accuracy of each model over the sampled frames from that chunk. Like other VDBMS, we define accuracy of a given model m based on the consensus between m and the reference model (typically also the most computationally-expensive model).

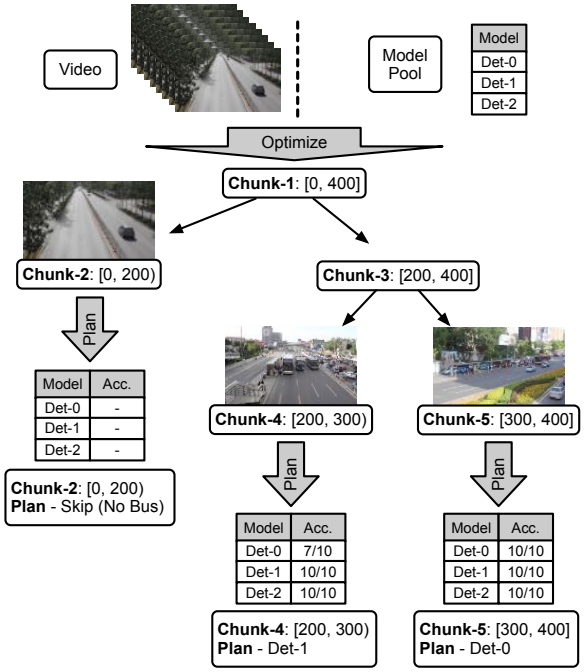


Figure 3: Fine-Grained Optimization over a Model Ensemble – Illustration of how FiGO processes the query in Listing 1.1.

In the example shown in Figure 3, the Optimizer first splits the video into two chunks: [0, 200] and [200, 400]. The Optimizer next determines to further split the [200, 400] into two chunks, based on the sample size bound and the cost model (elaborated in §4). The profiling results for each chunk are shown in the tables. For example, in the table associated with chunk-4, the accuracy of Det-0 is $\frac{7}{10}$ (*i.e.,* it agrees with Det-2 on seven out of ten frames). Based on collected metrics, the Optimizer constructs the query plan.

Assume that the [0, 200] chunk does not contain any bus. So, all the models do not find a bus object in any of the sampled frames. Then the Optimizer decides to completely skip processing the other 190 frames in this chunk. Assume that the [200, 300] chunk contains a bus in the background that is difficult to detect. For this chunk, the accuracy of the Det-0 model is lower than that of Det-1 and Det-2. Det-1 provides the same accuracy as Det-2 but is faster than Det-2. So, the Optimizer picks Det-1 for processing the remaining frames in this chunk. Assume that the [300, 400] chunk contains a bus in the foreground that is easy to detect. For this chunk, Det-0 is sufficient to accurately detect the existence of the bus. Hence, the Optimizer picks Det-0 for processing the rest of this chunk.

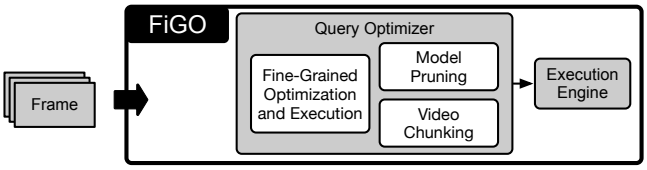


Figure 4: FiGO – System architecture of FiGO.

Theoretical Cost. We now provide a theoretical analysis of the benefits of fine-grained optimization over other approaches by quantifying only the execution time without the optimization overhead. To simplify our analysis, we assume that the VDBMS only uses two models $m1$ and $m2$. Let us denote the cost of running these models by C_{m1} and C_{m2}, respectively (where $C_{m1} < C_{m2}$). In the given video, let us denote the length of video that does not contain any positive events by α. Let the length of video that may be correctly processed by either model be β. Let the length of video that may only be correctly processed by the most accurate model $m2$ be γ. Then, the execution time of FiGO to process this video is:

$$\beta C_{m1} + \gamma C_{m2}$$

This is because the Optimizer skips chunks that are unlikely to contain positive events. In addition, it picks $m1$ to process the chunks that are easier to analyze.

With the model specialization approach, we only obtain the execution cost of the filtering technique since it is hard to guarantee the accuracy of a specialized model that directly answers the given query (*i.e.,* MS-Skip). The theoretical execution time of VDBMS that uses a specialized model for filtering out irrelevant frames is:

$$\beta C_{m2} + \gamma C_{m2} + \epsilon$$

This is because MS-Filter is likely to use $m2$ to process the entire video to obtain higher accuracy. The specialized model in front of $m2$ adds computational overhead (ϵ). The value of ϵ depends on the specialized model. We observe that this cost is already higher than that of FiGO as this approach does not leverage both models.

With the model cascade approach, the execution cost is:

$$\alpha C_{m1} + \beta C_{m1} + \gamma (C_{m1} + C_{m2})$$

This is because a VDBMS using a model cascade approach coupled with coarse-grained optimization never skips processing a chunk. Furthermore, with a model cascade, the VDBMS always runs the faster model before running the slower model, which increases the execution time associated with the γ subset of the given video.

3.2 System Architecture

We now present an overview of the system architecture of FiGO. As shown in Figure 4, the Optimizer is responsible for chunking the video and pruning the ensemble to lower optimization time.

Video Chunking. Video chunking consists of two parts: (1) required sample size estimation, and (2) cost estimation. The Optimizer must pick the optimal model to process a given chunk. By estimating the number of required samples, the Optimizer determines whether it must further split the chunk based on the given sample size bound. A query plan may be sub-optimal with respect to query execution time even if it meets the target accuracy. The Optimizer relies on a cost model to estimate the execution cost of a plan (*i.e.,* the selected model for a particular chunk). If the execution cost is higher than cost of additional chunking, the Optimizer further splits the chunk to lower the query execution cost.

Model Ensemble Pruning. If the Optimizer profiles all the models in the ensemble over the sampled frames in a chunk, the optimization overhead may outweigh the benefits of reduced query execution time. FiGO uses an online ensemble pruning technique that is based on Thompson sampling. This allows the Optimizer to only consider a smaller subset of models in the ensemble that have higher utility, thereby lowering the query optimization time.

4 VIDEO CHUNKING

In this section, we first motivate the need for chunking in §4.1. We then present a theoretical analysis of the accuracy of a given query plan based on the number of samples in §4.2. Lastly, we present a cost model for estimating the execution cost of a query plan in §4.3.

4.1 Variable Chunk Size

In FiGO, the Optimizer picks a model for processing a chunk by first evaluating a set of models over the sampled frames from the chunk. This profiling step is important to ensure the runtime performance and accuracy of FiGO. First, if a query is evaluated over an *insufficient* number of sampled frames from a chunk, then the query accuracy with respect to that chunk may be low. This is because the Optimizer may miss positive events that do not show up in the sampled frames (but are present in remaining frames of the chunk). Second, increasing the number of sampled frames leads to a higher query optimization time due to the profiling cost. If the plan is nearly optimal, and the VDBMS continues to collect more samples by further splitting the chunk, then it increases the overall query processing time.

To overcome these problems, FiGO splits the video into a sequence of differently-sized chunks. Since FiGO uniformly picks a fixed number (λ) of samples from each chunk, this approach is equivalent to varying the sampling rate across the video. A larger chunk size maps to a lower sampling rate. In contrast, a smaller-sized chunk maps to a higher sampling rate. During optimization, λ is always fixed. FiGO instead varies each individual chunk size to strike a balance between accuracy and query execution time. λ is a configurable parameter.

Figure 5 illustrates the chunking process. The Optimizer first treats the entire video as a single chunk. If the number of samples (λ) obtained from a given chunk c is lower than the estimated number of required samples, or the Optimizer determines that c may be further split using the cost model, then it continues to further split c. We discuss how the Optimizer estimates the lower bound of the number of required samples and how it uses the cost model in §4.2 and §4.3, respectively. Once the Optimizer determines that the actual sample size (λ) is higher than the estimated lower bound, and that the plan cannot be further optimized, it picks the model for processing the remaining frames within c.

4.2 Sample Size Lower Bound

We now explain how the Optimizer determines if the chunk is split enough to meet the accuracy constraint. Our key idea is

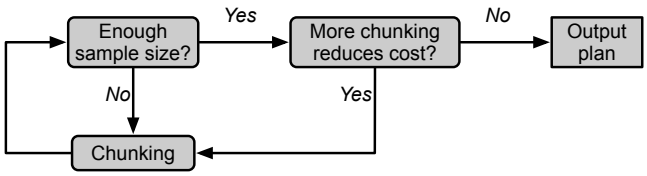


Figure 5: Video Chunking – Illustration of how the Optimizer chunks a given video in FiGO.

that, given the configurable sample size for each chunk λ, given an accuracy threshold $\mathbb{A}$, and an acceptable error range t, we estimate the probability that the accuracy of query execution plan is within the acceptable error range based on the observed average accuracy and the average accuracy deviations across *all* the models in the ensemble. We then use the probability to obtain a lower bound of sample size K [3]. Using the *sample size lower bound*, the Optimizer determines whether it needs to further split the chunk, so that actual sample size λ surpass the estimated lower bound K.

Proof Sketch. We now present a theoretical *sample size lower bound* of a chunk. To derive the bound, we first obtain the generalization error bound for a specific model m. We then obtain a probability bound for all the possible plans for a given accuracy threshold. By combining those two bounds, we compute the sample size lower bound.

Assume that the original video V is divided into R chunks.

$$V \rightarrow V_1\, V_2 \ldots V_R$$

We assume that the Optimizer needs at least K samples for each chunk, so that the Optimizer is able to derive a query plan with small error. So, it picks a total of KR samples. We denote the collected samples of the i^{th} chunk by $\hat{V}_i$. We assume that the Optimizer has a set of models in the ensemble (denoted by M). The Optimizer evaluates model m on frame x to obtain an accuracy metric denoted by $f(x, m)$. $f(x, m)$ compares results with respect to the reference model. If they are in consensus, it is set to 1. Otherwise, it is set to 0.

For a given model m, $f(x, m)$ over sampled frames is likely to differ from that computed across the entire video. This error is given by:

$$\mathop{\mathbb{E}}_{x\in\hat{V}} f(x,m) - \mathop{\mathbb{E}}_{x\in V} f(x,m)$$

To facilitate the computation of sample size lower bound, the average variance is given by:

$$\sigma^2 = \mathop{\mathbb{E}}_{x\in\hat{V}\, m'\in M} \Big[f(x,m') - \mathop{\mathbb{E}}_{m''\in M} f(x,m'')\Big]^2$$

It measures the average variance between the accuracy of a model and the mean accuracy of all models in the ensemble. Given an error threshold t, we next obtain the probability that error is within t using Bernstein's inequality [26]:

$$\mathbb{P}\Big[\mathop{\mathbb{E}}_{x\in\hat{V}} f(x,m) - \mathop{\mathbb{E}}_{x\in V} f(x,m) > t\Big] \le exp\Big(\frac{KRt^2}{2\sigma^2 + 2t/3}\Big)$$

[3]Like other VDBMSs, FiGO computes accuracy based on consensus with respect to the reference model. This metric is highly correlated with a canonical F-1 score.

Next, we seek to bound the probability that a particular plan returned by the Optimizer has a better accuracy than the accuracy threshold $\mathbb{A}$. Let us first define μ to be the mean accuracy of all models from the model ensemble over the all sampled video frames.

$$\mu = \mathop{\mathbb{E}}_{x\in\hat{V}, m'\in M} f(x,m')$$

The bound of a model m that has greater than $\mathbb{A}$ accuracy is:

$$\begin{aligned}
&\mathbb{P}\Big[\mathop{\mathbb{E}}_{x\in\hat{V}} f(x,m) > \mathbb{A}\Big] \\
&= \mathbb{P}\Big[\mathop{\mathbb{E}}_{x\in\hat{V}} f(x,m) - \mathop{\mathbb{E}}_{x\in\hat{V}, m'\in M} f(x,m') > \mathbb{A} - \mathop{\mathbb{E}}_{x\in\hat{V}, m'\in M} f(x,m')\Big] \\
&= \mathbb{P}\Big[\mathop{\mathbb{E}}_{x\in\hat{V}} f(x,m) - \mathop{\mathbb{E}}_{x\in\hat{V}, m'\in M} f(x,m') > \mathbb{A} - \mu\Big]
\end{aligned}$$

We next use Bernstein's inequality again to bound the term on right-hand-side to obtain:

$$\mathbb{P}\Big[\mathop{\mathbb{E}}_{x\in\hat{V}} f(x,m) > \mathbb{A}\Big] \le exp\Big(-\frac{KR(\mathbb{A}-\mu)^2}{2\sigma^2 + 2(\mathbb{A}-\mu)/3}\Big)$$

This represents the probability that a plan with a specific m has greater than $\mathbb{A}$ accuracy. Since there are $|M|$ models in the ensemble and R chunks in the video, the total number of possible plans is $|M|^R$. Using the previous bound, we obtain a bound on the number of execution plans that have accuracy greater than $\mathbb{A}$:

$$\Big|\Big\{m' \in M : \mathop{\mathbb{E}}_{x\in\hat{V}} f(x,m') > \mathbb{A}\Big\}\Big| \le |M|^R exp\Big(-\frac{KR(\mathbb{A}-\mu)^2}{2\sigma^2 + 2(\mathbb{A}-\mu)/3}\Big)$$

So far, we have obtained the bound that error of a plan is within t. We have also obtained the bound for number of plans that have accuracy better than $\mathbb{A}$. By combining of these two bounds, we obtain a bound on the sample size K such that plans with accuracy better than $\mathbb{A}$ have error within t. We also simplify the equation by Jensen's inequality [11]:

$$\begin{aligned}
&\mathbb{P}\Big[\mathop{\mathbb{E}}_{x\in\hat{V}} f(x,m) - \mathop{\mathbb{E}}_{x\in V} f(x,m) > t\Big], \forall m \in M \\
&\qquad \le |M|^R exp\Big(-\frac{KR(\mathbb{A}-\mu)^2}{2\sigma^2 + 2(\mathbb{A}-\mu)/3}\Big) exp\Big(-\frac{KRt^2}{2\sigma^2 + 2t/3}\Big) \\
&\qquad \le |M|^R exp\Big(-\frac{KR(\mathbb{A}-\mu)^2}{2\sigma^2 + 2(\mathbb{A}-\mu)/3} - \frac{KRt^2}{2\sigma^2 + 2t/3}\Big) \\
&\qquad \le |M|^R exp\Big(-\frac{KR(t+\mathbb{A}-\mu)^2}{4\sigma^2 + 2(t+\mathbb{A}-\mu)/3}\Big)
\end{aligned}$$

We expect the value to reach 0 as R grows (*i.e.,* in the extreme case, generalization error should be 0 if query is optimized for every frame or chunk size is 1). To achieve that, the base should be less than 1 as expressed below:

$$\begin{aligned}
1 &\ge |M| exp\Big(-\frac{K(t+\mathbb{A}-\mu)^2}{4\sigma^2 + 2(t+\mathbb{A}-\mu)/3}\Big) \\
\frac{K(t+\mathbb{A}-\mu)^2}{4\sigma^2 + 2(t+\mathbb{A}-\mu)/3} &\ge log(|M|) \\
\lambda \overset{?}{\ge} K &\ge \frac{log(|M|)(4\sigma^2 + 2(t+\mathbb{A}-\mu)/3)}{(t+\mathbb{A}-\mu)^2}
\end{aligned}$$

Thus, given a desired error range t and expected accuracy $\mathbb{A}$, the Optimizer calculates sample size lower bound K using the observed

variance between models and the mean model accuracy. If actual sample size λ in a chunk exceeds the calculated K, the Optimizer stops further optimization unless forced by the cost model.

4.3 Cost Model

Lastly, we discuss how the Optimizer uses the cost model. The idea is to compare the cost of further splitting the chunk (*i.e.,* more optimization time) against the estimated reduction in query execution time due to more fine-grained plans. If the estimated reduction in query execution time outweighs the additional optimization overhead, then the Optimizer proceeds to further split the chunk.

We model the execution cost of a specific chunk as the product of the cost of the assigned model C_m and number of frames in a specific chunk $|V_i|$:

$$|V_i|C_m$$

Thus, the total cost of splitting the chunk is given by the sum of additional optimization cost and the cost of executing the new plan. Since splitting a chunk into two sub-chunks leads to doubling the number of samples, the optimization cost is given by the cost of evaluating all models in a collection M over the additional samples (λ samples for each chunk):

$$2\lambda \sum_{m' \in M} C_{m'} - \lambda \sum_{m' \in M} C_{m'}$$

To estimate the execution cost of the new plan, the challenge here is that which models will be selected for the sub-chunks is unknown. However, we may obtain a lower bound on the execution cost. We denote the estimated cost of model profiling for left and right sub-chunks by C_{m_l} and C_{m_r}, respectively. The models found by next iteration of optimization to use for the new sub-chunks should be at least as good as the model for left and right sub-chunks found during current iteration for providing good accuracy. Their costs are denoted by $C_{m_{lc}}$ and $C_{m_{rc}}$. So C_{m_l} and C_{m_r} must be greater than or equal to $C_{m_{lc}}$ and $C_{m_{rc}}$ (more expensive or slower model provides better accuracy). Since the size of the new sub-chunk is half of that of the original chunk, we estimate execution cost by calculating its lower bound:

$$\frac{|V_i|}{2}C_{m_l} + \frac{|V_i|}{2}C_{m_r} = \frac{|V_i|}{2}C_{m_{lc}} + \frac{|V_i|}{2}C_{m_{rc}}$$

So, the potential reduction in query execution time is given by:

$$\frac{|V_i|}{2}C_{m_l} + \frac{|V_i|}{2}C_{m_r} - |V_i|C_m$$

To justify the optimization overhead, the reduction in execution time must be higher than the increase in optimization time:

$$\lambda \sum_{m' \in M} C_{m'} \leq \frac{|V_i|}{2}C_{m_l} + \frac{|V_i|}{2}C_{m_r} - |V_i|C_m$$

The Optimizer computes this bound to strike a balance between query execution time and optimization time. If the estimated increase in optimization time is justified, it continues to split the chunk even if K is less than λ frames.

5 ONLINE MODEL ENSEMBLE PRUNING

In this section, we present the model ensemble pruning technique that Optimizer uses to lower the optimization overhead. We first motivate the need for ensemble pruning and its connection to multi-bandit problem and Thompson sampling [29] in §5.1 and §5.2. We conclude with a discussion on how we tailor the Thompson sampling algorithm to the ensemble pruning problem in §5.3.

5.1 Motivation

FiGO uses a collection of models to accelerate queries. In particular, the ensemble consists of eight object detection models from the EfficientDet family [28]. Each model offers a unique trade-off between the accuracy and query execution time. To maximize speedup, FiGO supports as many models as possible. However, during the profiling step, the Optimizer spends a significant amount of time evaluating all the models over the sampled frames. To tackle this problem, we leverage the observation that even though different models are used across different videos, it is often the case that only a subset of models is used while processing a given video. Thus, the Optimizer prunes out the subset of models that is not likely to be used for a given query or video dataset, thereby lowering the overhead of the profiling step.

5.2 Connection to Multi-Armed Bandit

In reinforcement learning, the multi-armed bandit problem demonstrates [15] the dilemma between exploration and exploitation [7, 20]. A player has access to a set of slot machines and the goal is to maximize gain by playing on these machines. The dilemma lies in whether: (1) the player should try out other slot machines to discover machines with higher reward (exploration), or (2) the player should stick to playing on a certain machine (exploitation).

The Optimizer faces a similar challenge while pruning the model ensemble. To identify which models to prune, the Optimizer must evaluate them on the sampled frames during profiling. While this will eventually lead to better ensemble pruning, the profiling overhead may outweigh those benefits. On the other hand, if the optimizer only evaluates a subset of models that delivers high accuracy, that would lead to imperfect model pruning as the Optimizer is unable to leverage faster, less accurate models.

Thompson Sampling. This is a widely-used technique [1, 5] for solving the multi-armed bandit problem. It begins with no assumption about the reward of different actions. It then refines the estimation of reward associated with each action by exploring the action space. A key benefit of this technique is that it returns the reward estimate along with a confidence score. We tailor this algorithm to tackle the ensemble pruning problem in FiGO.

5.3 Ensemble Pruning via Thompson Sampling

The goal of the Optimizer is to find the performant model that makes the most correct predictions and discard other models. We may view the process of evaluating a specific model as an action a. The reward associated with a model Q_a is given by the average number of correct predictions made by that model. Thus, we formulate the ensemble pruning problem as finding the model among all models that maximizes the number of correct predictions (Q).

$$\underset{a \in A}{argmax}(Q_a)$$

In reality, the Q may only be estimated because the VDBMS does not know how a model performs a priori. The Optimizer can estimate

Q at time t by using the average of the observed reward value R [4] across all the sampled video frames examined before t.

$$Q_t(a) = \frac{1}{n}\sum_{i=1}^{n} R_i$$

However, this greedy algorithm requires the Optimizer to evaluate every model for the same number of times, leading to a higher optimization overhead.

To overcome this problem, we tailor the Thompson sampling algorithm to estimate the average number of correct predictions of each model. The estimator allows the Optimizer to quickly converge to evaluating only a smaller number of models. As Thompson sampling, the estimator uses three variables: (1) n records how many times a model is evaluated. (2) τ represents the confidence of the estimation (higher confidence level if a model is evaluated more times), so it is correlated to n. (3) μ represents the expected value of estimated reward $E[Q]$. For each model, the estimated reward Q is updated as a running average. The Optimizer updates the expected value of estimated reward weighted on variable τ as shown below:

$$Q_{t+1} = \left(1 - \frac{1}{n}\right)Q_t + \frac{1}{n}R$$
$$\mu_{t+1} = \frac{\tau_t \mu_t + nQ_{t+1}}{\tau_t + n}$$

When the Optimizer checks the expected reward of a model, at time t, instead of returning Q directly, it returns a random value that is sampled from a probability distribution based on μ and τ:

$$Q_t \sim \mathcal{N}\left(\mu_t, \sqrt{\frac{1}{\tau_t}}\right)$$

This distribution not only accounts for the encountered values of Q, but also incorporates confidence of the evaluation.

Performance Cost. Typically, in the multi-armed bandit problem, the system only cares about an action with the highest reward. However, in FiGO, the Optimizer needs to consider both accuracy and execution cost of a particular model. So, we include the execution cost of a model C_a in the estimator. We use the profiled inference time per frame of a model as its execution cost.

$$\underset{a\in A}{argmin}(C_a) \wedge \underset{a\in A}{argmax}(Q_a)$$

This ensures that if two models have similar accuracy, then the Optimizer picks the faster model.

Exploration Expansion. The canonical Thompson sampling algorithm only updates the estimated reward value of the best action. However, in FiGO, pruning all models in the model ensemble to one model does not minimize the query processing time. We empirically find that pruning the ensemble to three models provides sufficient flexibility for the Optimizer. Pruning to fewer models lowers accuracy, and pruning to more models increases optimization overhead. So, we configure the Optimizer to only consider the top three models during query optimization. However, for the pruning algorithm, Optimizer must initially evaluate *all* the models over the first chunk to determine their accuracy and inference time metrics. For those selected models, the estimator evaluates all of them on sampled frames and updates their reward.

[4] In FiGO, the observed reward value is 1 if the model is in consensus with the reference model. Otherwise, it is 0.

Algorithm 1: Query optimization.

Input : **V** – Video.
M – Model ensemble (*e.g.,* 8 models from EfficientDet).
λ – Sample size for each chunk (*e.g.,* 10 frames).
$\mathbb{A}$ – User-specified query accuracy (*e.g.,* 0.95).
t – Tolerable error bound (*e.g.,* 0.03).
E – Estimator used for model pruning.
Output : Return a list of execution plans.

1 **return** `GetQueryPlan(`*0,* `Length(`**V**`))`

2 **Function** `GetQueryPlan(`*start, end*`)`
  Output : A collection of plans under $\mathbb{A}$ constraint within t.
3 $V_i \leftarrow$ **V** [start, end] `// Obtain chunk.`
4 $\hat{V} \leftarrow$ `UniformSample(`V_i, λ`)` `// Sample frames.`
5 $\hat{M} \leftarrow$ `PruneModel(`**E**, **M**`)`
6 R $\leftarrow$ {}
  `// Profiling step.`
7 **for** $v \in \hat{V}$ **do**
8 **for** $m \in \hat{M}$ **do**
9 R += { [v, m]: `Predict(`v, m`) ==`
10 `Predict(`v, $m_{reference}$`)` }
11 `UpdateEstimator(`**E**, R, $\hat{M}$`)`
12 K $\leftarrow$ `EstimateSampleSize(`R, $\hat{M}$`)`
13 FurtherChunkingCost, ExecCost $\leftarrow$ `EstimateCost(`R, $\hat{M}$`)`
  `// Determine whether to continue splitting.`
14 plan $\leftarrow$ {}
15 **if** *(*$K \leq \lambda$ **and** *ExecCost < FurtherChunkingCost)*
16 **or** *(* $|end - start| \leq 100$ *frames)* **then**
17 plan += { [start,end]: `PickBestModel(`R, $\hat{M}$, $\mathbb{A}$, t`)` }
18 **else**
19 plan += `GetQueryPlan(`*start,* $\frac{end}{2}$`)`
20 plan += `GetQueryPlan(`$\frac{end}{2}$, *end*`)`
21 **return** plan

Ensemble Pruning + Video Chunking. We adapt the estimator to operate on fine-grained chunks. First, it is important to let the Optimizer do sufficient exploration at very beginning. So, the Optimizer evaluates all the models on the very first chunk. Otherwise, it may prune certain models without sufficient analysis. Second, the Optimizer prunes and updates the estimated reward values for models at chunk granularity. For a given chunk, the Optimizer first decides which models to prune based on their estimated reward value. It then evaluates those models to construct the plan. Based on the profiling step, it updates the estimated reward value of the evaluated models. For the next chunk, the Optimizer prunes the models based on their updated reward estimates. This enables fine-grained, local adaptation of ensemble pruning to each chunk.

6 QUERY OPTIMIZATION ALGORITHM

In this section, we present the overall query optimization algorithm. As shown in Algorithm 1, the Optimizer operates on a chunk V_i defined by the start and end frames. It begins by treating the entire video V as a single chunk (Line 2). Given a chunk c, the Optimizer

picks a λ number of samples from c (Line 4). The Optimizer then prunes the set of models using the estimator that is built using a variant of Thompson sampling (Line 5). For the very first chunk, it does not prune any model. For later chunks, it prunes the ensemble to three models.

Next, the Optimizer computes reward by measuring consensus between a model and the reference model on sampled video frames. It then updates the reward estimates (Line 11). Using the techniques presented in §4, the Optimizer estimates the required sample size bound (K) and the benefits of further splitting the chunk based on the current query plan. If the number of samples (λ) is greater than the lower bound K, and further splitting does not lower the query processing time, then the Optimizer returns the current plan (Line 16). The Optimizer also stops searching for a better plan even when the chunk is too small (*i.e.,* less than 100 frames). Otherwise, the Optimizer continues to recursively split the chunk and compute plans for the resulting sub-chunks (Line 20). We next present two additional optimizations to reduce the query processing time:

Affinity-based Sampling for Resusing Results. Evaluating the consensus between models (Line 10) is computationally expensive as the VDBMS must evaluate those models. We use an affinity-based sampling technique that enables the Optimizer to reuse profiling results. In Line 4, instead of using a strict uniform random sampling algorithm, the Optimizer prioritizes frames over which other models have already been evaluated. This allows the Optimizer to reuse profiling results, thereby reducing the number of model invocations.

Chunk Size Limit. As shown in Line 16, we constrain the size of the smallest chunk to 100 frames. This corresponds to a maximal sampling rate of 10% (λ is set to 10 frames based on a sensitivity analysis shown in §7.7). The reasons are two-fold. First, we configure the other baselines we compare FiGO against to use 10% of the video frames. To ensure a fair comparison, we also constrain the maximal sampling rate of FiGO. Second, we empirically found that this maximal sampling rate strikes a balance between accuracy and query processing time.

7 EXPERIMENTAL EVALUATION

We seek to answer the following questions in our evaluation:

RQ1 – How effective is FiGO compared to the state-of-the-art techniques for accelerating queries in VDBMSs?

RQ2 – How does FiGO perform on complex queries with multiple atomic predicates?

RQ3 – What is the impact of the accuracy threshold?

RQ4 – How much does pruning lower optimization time?

RQ5 – How optimal is the plan found by the Optimizer?

RQ6 – How does sample size affect accuracy & processing time?

RQ7 – How does FiGO generalize to another model ensemble?

7.1 Evaluation Setup

Evaluation metrics. Like other VDBMSs [12–14], we measure accuracy with respect to the reference model (*i.e.,* Det-7). We compute F-1 score of a baseline relative to the results of the reference model. We report query optimization time, execution time, and overall processing time (*i.e.,* optimization time + execution time).

Queries. Similar to the query shown in §1, we evaluate the baselines on queries that focus on finding frames containing target object(s). We vary the dataset and the predicate to construct these queries. We report the average selectivity of these queries in Table 2 (*i.e.,* the fraction of frames that satisfy the predicate(s)).

Evaluated Techniques. We reimplement these three key techniques used in state-of-the-art DBMSs: (1) model specialization with filtering (MS-Filter), (2) model cascade (MC), and (3) model ensemble with coarse-grained optimization (ME-Coarse).

Model Ensemble. FiGO uses the EfficientDet family of object detection models [28]. In particular, the ensemble consists of eight models (Det-0∼Det-7) with different accuracy-execution time tradeoffs. All the models are pre-trained on COCO dataset [17]. FiGO uses these off-the-shelf models to robustly answer diverse queries over different datasets. We configure the sample size of each chunk (λ) used by FiGO to 10 frames based on a sensitivity analysis (§7.7).

MS-Filter. With model specialization, we construct a baseline inspired by PP. The model configuration consists of a lightweight filter that discards irrelevant frames followed by the heavyweight, reference model. We configure MS-Filter to picks the optimal filter based on evaluation on 10% of the frames (elaborated in §7.7). It picks a filter from Det-0 through Det-6 models, and uses Det-7 as the reference model. We found that directly returning answers using lightweight model often returns inaccurate results (MS-Skip). So, we do not compare FiGO against this baseline.

MC. We replicate the model cascade approach by connecting all the eight models in EfficientDet in a sequence. We first optimize MC on 10% video frames to determine the optimal confidence thresholds to use for a given query. During query execution, MC decides to when to short-circuit the inference based on confidence threshold.

ME-Coarse. This baseline also uses a model ensemble similar to FiGO. Unlike FiGO, ME-Coarse takes a coarse-grained approach to optimization. It uniformly samples 10% of the frames from the entire video and profiles all the models over the sampled frames. It then picks exactly one model from the ensemble to process the remaining frames in the video.

Datasets. As shown in Table 2, we evaluate these baselines on four representative video datasets: (1) UA-DeTrac [31], (2) Jackson-Town dataset from [14], (3) a subset of BDD100K dataset [33], and (4) a subset of VIRAT dataset [23].

UA-DeTrac and Jackson-Town. These datasets are obtained from traffic surveillance cameras. The majority of the objects in their videos are vehicles: cars, trucks, or buses. These datasets differ in terms of video length and content. Videos in UA-DeTrac are relatively short. They are often only 30 seconds long (*i.e.,* 1K video frames). In contrast, videos in the Jackson-Town dataset are longer.

BDD. Unlike the previous dataset, the BDD dataset consists of vidoes obtained from dashcams. Since the videos are obtained from a *moving* camera, it is more challenging to deliver accurate answers on them. Videos in the BDD dataset are also comparatively short (∼ 1K frames). Besides vehicles, the videos contain many traffic lights.

Query	Dataset	Predicate	Avg. Video Size	Avg. Sel.	Chunk Size Avg.	Chunk Size Std.	Qry Proc. Time (s)	Qry Exec. Time (s)	Qry Opt. Time (s)
Q1.1	UA-DeTrac	`Count(Car)≥4`	1404	0.95	273	276	128.8	99.3	29.5
Q1.2	BDD	`Count(Traffic Light)≥1`	1205	0.47	163	69	187.1	149.4	37.7
Q1.3	VIRAT	`Count(Person)≥2`	1316	0.76	149	99	208.7	161.0	47.7
Q1.4	Jackson	`Count(Car)≥1`	10000	0.23	194	515	815.2	571.9	243.3
Q2.1	UA-DeTrac	`Count(Car)≥4 AND Count(Bus)≥1`	1404	0.73	166	150	186.3	141.6	44.7
Q2.2	BDD	`Count(Traffic Light)≥1 AND Count(Car)≥1`	1205	0.46	165	67	190.2	152.3	37.9

Table 2: Query Chacteristics – Properties of queries and their associated video datasets. We first report the average size of videos and the selectivity of the queries. We then present the average chunk size and standard deviation. We finally report the overall query processing, optimization, and execution time.

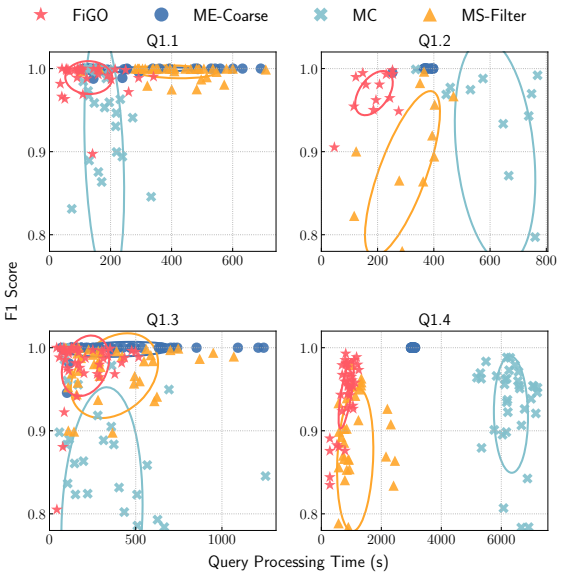


Figure 6: End-to-end Performance – F-1 and query processing time of all the system across four queries.

VIRAT. To ensure that FiGO works well on objects other than vehicles, we evaluate it on the VIRAT dataset for pedestrian detection. We pick a subset of videos from VIRAT that mainly contains difficult-to-detect human objects in the background of the frames. These videos range from 400 to 3K frames.

Software and Hardware. We implement FiGO with the PyTorch [24] framework. We use a server with 44 CPU cores and 256 GB memory along with one Titan Xp GPU with 12 GB memory.

7.2 RQ1 – End-to-End Performance

In this experiment, we first compare the F-1 score and query processing time of all the systems. The results are shown in Figure 6. With FiGO, we set the accuracy threshold α to 0.95 and the error range t to be 0.03. We evaluate the query on the target video dataset five times and take the average of collect the metrics. We separately plot the F-1 score and query processing time on each video. We also mark the centroid across all videos for a given system using a scatter plot. The most notable observation is that FiGO outperforms other systems on both accuracy and query processing time.

Q1.1. The UA-DeTrac dataset contains videos of busy traffic intersections. So the selectivity of this query is high, as shown in Table 2. Due to this, MS-Filter has very high query processing time demonstrated in Figure 6. In contrast, both ME-Coarse and MC accelerate the query on some videos. While ME-Coarse offers a high F-1 score as it is easier to optimize, it leads to higher query processing time (*e.g.,* 632 s) on videos that require more compute-intensive models. MC is faster but its F-1 score is lower (*e.g.,* it drops to 0.82 on a video). Compared to other systems, FiGO delivers better F-1 and processing time. We attribute this to two factors. First, FiGO finds better fine-grained plans that often use faster models. Second, FiGO uses a sample size bound and ensemble pruning to lower optimization time, that leads to faster query processing.

Q1.2 and Q1.3. Besides vehicles, we compare the performance of FiGO against other baselines on more complex objects like traffic lights and pedestrians. Since these objects are relatively smaller in the frame, the VDBMS often requires slower, compute-intensive models to provide accurate predictions. As shown in Figure 6, FiGO still consistently beats ME-Coarse and MC on both queries (1.8× and 3.2× on Q1.2 and 1.9× and 1.4× on Q1.3). MS-Filter is comparable to FiGO, as its filter is able to discard many irrelevant video frames on these low selectivity queries. FiGO is 1.5× and 1.2× faster than MS-Filter and delivers 0.06 and 0.01 higher average F-1 score on Q1.2 and Q1.3, respectively.

Q1.4. Lastly, we compare FiGO against other baselines on a query that operates on much longer videos. In this case, the processing time difference between FiGO and other systems is more prominent. This illustrates the importance of using fine-grained optimization in tandem with a model ensemble. MS-Filter also has lower query processing time as the filter is effective in this low query selectivity (Table 2). In this case, FiGO still shows is 1.4× faster than MS-Filter and delivers 0.08 higher F-1 score on average across all videos.

Chunk Size. We report the average chunk size and the standard deviation of chunk size for each query when the videos are processed by FiGO. The results indicate that the Optimizer is effectively adjusting the chunk size based on the queries and the contents of the videos. For instance, the average chunk size decreases from 299 frames in Q1.1. to 243 frames in Q2.1 due to harder predicate.

7.3 RQ2 – Complex Queries

In this experiment, we examine the performance of FiGO and other systems on two representative complex queries with multiple

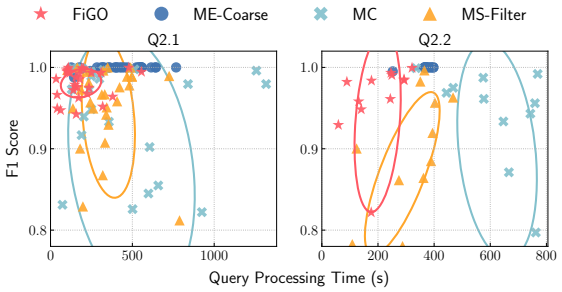


Figure 7: End-to-end Performance on Complex Queries – F-1 and query processing time of all the system across two complex queries.

	Query	F-1 Score	Qry Proc. Time (s)	Qry Exec. Time (s)	Qry Opt. Time (s)
ME-Coarse-Join	Q2.1	0.99	691.69	495.38	196.31
	Q2.2	0.99	588.30	435.52	152.78
FiGO-Join	Q2.1	0.99	295.17	242.19	52.98
	Q2.2	0.99	275.88	238.66	37.22
FiGO	Q2.1	0.98	186.31	141.69	44.62
	Q2.2	0.95	190.23	152.36	37.87

Table 3: FiGO *vs.* FiGO – Join – F-1 score and query processing time metrics of: (1) FiGO (evaluating predicates together) and (2) FiGO – Join (evaluating predicates separately).

predicates (Q2.1 and Q2.2 in Table 2). For example, the relational algebraic plan for Q2.1 is shown below:

$$\sigma_{Classification < \text{Count(Car)} \geq 4 \wedge \text{Count(Bus)} \geq 1 >}(\text{UA-DeTrac})$$

We tailor the Optimizer to find a model that works well for *both* predicates together.

As shown in Figure 7, on both queries, FiGO delivers a lower query processing time than other systems. As the predicates become more complex, MC requires significantly longer query processing time than others. In case of MS-Filter, complex predicates lead to lower selectivity. So the filter is able to discard more frames using faster models. However, we observe that this approach suffers from lower F-1 score compared to FiGO (0.04 and 0.06 points on Q2.1 and Q2.2, respectively). ME-Coarse also suffers from higher query processing time since it does not leverage intra-video opportunities.

Execution with Join. We next compare the current technique for jointly evaluating a complex predicate against another technique that *separately* evaluates the predicates and joins the results based on the frame number. With this technique, the algebraic plan for Q2.1 is given by:

$$(\sigma_{Classification < \text{Count(Bus)} \geq 1 >}(\text{UA-DeTrac})) \bowtie_{\text{frameID}} (\sigma_{Classification < \text{Count(Car)} \geq 4 >}(\text{UA-DeTrac}))$$

In this case, the Optimizer in FiGO separately finds the optimal model for each atomic predicate. After obtaining results from both models, it joins them to return the final set of frames that satisfies both atomic predicates.

We compare these two techniques on Q2.1 and Q2.2. We also implement join with ME-Coarse for comparison (as it is the strongest

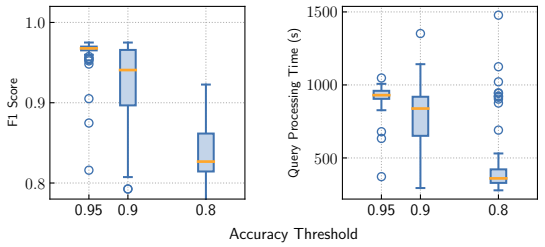


Figure 8: Impact of Accuracy Threshold – F-1 score and query processing time metrics under different accuracy thresholds.

baseline). We report the average F-1 score and query processing time metrics associated with these systems in Table 3. Our results demonstrate that there is a tradeoff between these two approaches. First, evaluating each predicate separately improves F-1 score. This is because combining multiple predicates together complicates the process of estimating the sample size bound. In contrast, evaluating multiple predicates together in the Optimizer lowers query processing time (due to lower optimization time and execution time). Better support for complex predicates is beyond the scope of this paper. We plan to explore this problem in the future.

7.4 RQ3 – Impact of Accuracy Threshold

In this experiment, we investigate how the accuracy threshold affects the F-1 score and query processing time. We use Q1.4 that operates on longer videos. This allows us to better illustrate the impact of the accuracy threshold and the associated sample size bound. We evaluate FiGO across three accuracy thresholds: 0.95, 0.9, and 0.8. Across all cases, we configure the tolerable error range t to 0.03. FiGO executes Q1.4 100 times. We report the min, 25% percentile, median, 75% percentile, and max of results of all executions.

As shown in Figure 8, when we lower the accuracy threshold, the final query F-1 score is reduced as expected. This is because the sample size bound computed by the Optimizer is highly correlated with the final F-1 score. We note that FiGO does not provide a strict accuracy guarantee. In addition, we observe that the variance of F-1 score increases when accuracy threshold is lowered. This is because while the Optimizer may find a good plan with fewer samples, it often ends up with a sub-optimal plan.

Besides F-1 score, we also evaluate the query processing time of FiGO under different accuracy thresholds. We discover that speedup increases when the threshold is lowered. The reasons are two-fold. First, fewer samples are needed by the Optimizer when the system has a lower accuracy threshold, leading to lower optimization time. Second, with fewer samples, the Optimizer often picks a faster, less accurate model for processing the chunk. Thus, a sub-optimal plan also results in lower execution time.

7.5 RQ4 – Optimization Time

In this experiment, we examine the efficacy of model pruning in reducing the optimization time. We focus on three systems: ME-Coarse, FiGO - no pruning, and FiGO. FiGO - no pruning differs from FiGO in that it picks the best model from all the models in the ensemble. Similar to FiGO, it also chunks the video during optimization. We measure the optimization time of all the queries across these three systems. We report the average time spent on

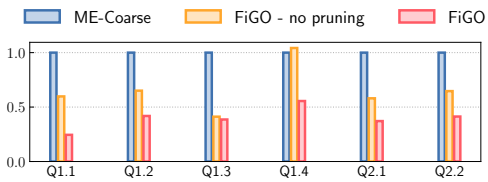


Figure 9: Optimization Time – Comparison of optimization overhead of three systems: ME-Coarse, FiGO without model pruning, and FiGO.

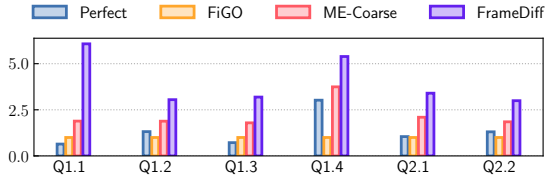


Figure 10: Plan Optimality – Comparison of query execution time with respect to that of the FiGO plan across four systems.

query optimization over five runs. We normalize the optimization time of each system against that of ME-Coarse.

The most notable observation shown in Figure 9 is that model pruning further reduces the optimization overhead. The reduction in optimization time depends on the query (*e.g.,* Q1.1 *vs.* Q1.3). This is because optimization time is unevenly distributed over different models (*i.e.,* evaluating a compute-intensive model costs more than evaluating a faster model). If the system prunes more compute-intensive models, the reduction in is more significant.

Q1.1. This query only requires a faster model. So FiGO prunes all the compute-intensive models. Thus, ensemble pruning lowers optimization time by 2× as shown in Figure 9.

Q1.3. Correctly detecting pedestrian objects in Q1.3 is much harder. This is only feasible with compute-intensive object detection models. So, on this query, FiGO only prunes the faster models (must keep the slower models to meet the accuracy constraint). So, the impact of pruning on optimization time is smaller than that in Q1.1.

Discussion. As listed in Table 2, optimization time accounts for ~25% of the total query processing time. So, the optimization overhead is non-trivial compared to the total processing time. To further reduce the optimization overhead, we consider training a *selector model* that is a fast deep neural network to directly estimate which model to use (instead of relying on the profiling step). Nevertheless, this approach suffers from two limitations. First, it is challenging to obtain enough training data. Second, the collected training dataset often has a skewed distribution. Due to its ineffectiveness, we do not include its results in this paper.

7.6 RQ5 – Plan Optimality

We refer to the plan that delivers the smallest query processing time with no loss in accuracy with respect to the reference model as the *perfect plan*. In this experiment, we study the optimality of plans generated by ME-Coarse, FiGO, and another baseline by comparing their query execution time against that of the corresponding perfect plan. We examine the FrameDiff baseline in this

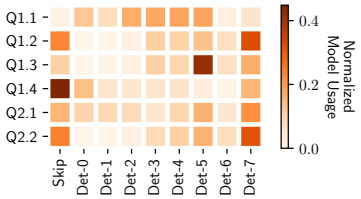


Figure 11: Model Usage Distribution – The distribution of usage of models in the ensemble across all queries.

experiment. FrameDiff leverages on the frame skipping technique presented in NoScope [14]. It uses traditional frame-wise structural similarity index to determine whether inference with the reference model is necessary for the given frame. We identify the perfect plan by profiling all the models over every chunk. While this leads to high optimization overhead, the goal of this experiment is to only compare the query execution time associated with the resulting plan. We normalize the execution time metrics of these systems against that of FiGO.

The results shown in Figure 10 demonstrate that the plans provided by ME-Coarse is sub-optimal compared to the corresponding perfect plans resulting in large performance gaps. FiGO consistently outperforms ME-Coarse. Execution time of FiGO may be further improved on Q1.1 and Q1.3. However, getting closer to the perfect plan will also increase the optimization overhead as it corresponds to smaller-sized chunks (*i.e.,* high sampling rate). On queries like Q1.2, FiGO even outperforms the perfect plan. This is because the Optimizer selects faster models than that required for accurate predictions leading to tolerable drop in F-1 score. FrameDiff has higher normalized execution time than other systems. The reasons are two-fold. First, the target objects may be very small with respect to the entire frame. In this case, frame difference calculated by traditional algorithm is not able to detect any significant change in the frame. Second, it must configure the threshold value used to determine whether frame difference is significant enough to require inference with the reference model. To ensure tolerable accuracy, we configure a conservative threshold value, that further increases the execution overhead.

Model Usage Distribution. Lastly, we examine the distribution of usage of models in the ensemble across all queries. The results are shown in Figure 11. The distribution changes across queries that operate on different datasets. On queries with low selectivity (*e.g.,* Q1.4), the Optimizer skips several chunks. On queries with hard-to-detect events (*e.g.,* Q1.2 and Q2.2), it frequently uses the reference model to reach a high F-1 score. This illustrates that Optimizer adapts the plan to different queries and datasets.

7.7 RQ6 – Impact of Sampling Rate

In this experiment, we investigate the impact of sampling rate. We examine these settings for sampling rate: 2%, 5%, 10%, 20% and 50%. Recall that FiGO always picks fixed number (λ) of samples from each chunk but varies the size of each chunk. We instead vary the value of λ: 2, 5, 10, 20, and 50 samples in each chunk.

The results for a representative query (Q1.3) are shown in Figure 12. With FiGO, if very small number of samples are picked from each chunk, then the sample size lower bound does *not* work due

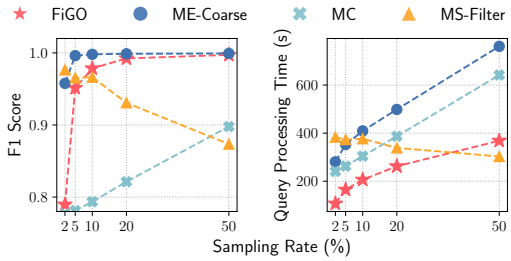
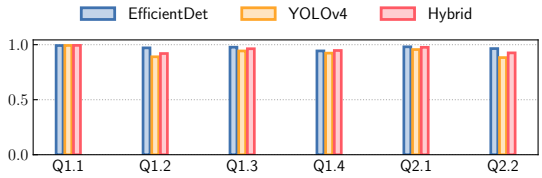
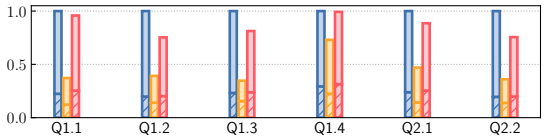


Figure 12: Impact of Sample Rate – Comparison of the impact of sample rate on: (1) F-1 score, and (2) Query processing time.



(a) Generalization to Other Ensembles: F-1 Score – Comparison of F-1 score between different model ensembles.



(b) Generalization to Other Ensembles: Processing Time – Comparison of processing time between different model ensembles. We also report optimization time (hatched pattern) vs execution time breakdown.

Figure 13: Generalization to Other Ensembles – FiGO on different model ensembles.

to high variance. So, this leads to lower F-1 score ($\lambda == 2$). With ME-Coarse, sampling rate has significant impact on both F-1 score and query processing time. It lowers processing time by lowering the sampling rate, but that also lowers the F-1 score. F-1 score of ME-Coarse does not improve when it surpass 10% sampling rate. FiGO delivers lower query processing time under the same F-1 score. With MS-Filter, a higher sampling rate causes lower F-1 score. The reason is because in our setting, MS-Filter is able to pick a more aggressive filtering threshold value, which can incorrectly filter out positive events. Based on these results, we configure the sampling rate to 10% in all the other experiments to do a fair comparison between systems. While other systems may also reduce sampling rate to lower query processing time (*e.g.,* 5% for ME-Coarse), FiGO is still significantly faster.

7.8 RQ7 – Generalization to Other Ensembles

We next study the ability of FiGO to generalize to other model ensembles. We first evaluate FiGO over the Scaled-YOLOv4 ensemble [30] with five object detection models. The models in the Scaled-YOLOv4 ensemble deliver similar accuracy to those in EfficientDet. They support faster inference, but only offer a few performance-accuracy tradeoffs. We also evaluate FiGO over a hybrid ensemble (*i.e.,* Hybrid) that combines all the models in the EfficientDet and Scaled-YOLOv4 ensembles (13 models in total).

In Figure 13a, we report the average F-1 score comparison between three model ensembles. The F-1 score of all model ensembles is evaluated against the reference model in EfficientDet. In Figure 13b, we report the query processing time of three model ensembles. The processing time is normalized to the EfficientDet model ensemble. We report the time breakdown by showing the optimization time (hatched pattern) and execution time.

The most notable observation is that the idea of fine-grained query optimization generalizes to different model ensembles. FiGO delivers good F-1 scores with all the ensembles. Scaled-YOLOv4 ensemble has lower F-1 score compared to other ensembles, because of limited model diversity. So, it does not provide smooth tradeoffs between processing time and F-1 score. In the case of Hybrid, FiGO is able to pick an optimal query execution plan. The F-1 score of Hybrid is higher than Scaled-YOLOv4 ensemble, and it has lower query processing time than EfficientDet ensemble.

8 RELATED WORK

We present a brief review of related work on query optimization. DB2's LEarning Optimizer (Leo) was a pioneering effort in improving the efficacy of query optimizers [27]. Leo learns from its mistakes by adjusting its statistical estimates over time. More recently, Neo [21] adopts a novel deep neural network-based approach for finding the optimal plan. It uses a value network for accurately predicting the latency of partial and complete query plans.

The overhead of query optimization in production DBMSs has increased over time. Developers of MemSQL reported that optimization time in analytical workloads may even be higher than query execution time [6]. It is critical for the Optimizer to strike a balance between optimization and execution time. We discuss how FiGO tackles this problem in §5.

9 CONCLUSION

We presented, FiGO, a video analytics system for efficiently processing visual data at scale. FiGO couples a model ensemble approach with fine-grained query optimization. Its Optimizer first splits the video into a sequence of differently-sized chunks based on a sample size bound. It then picks a model from the ensemble that delivers the lowest query execution time while meeting the target accuracy constraint. Lastly, its Execution Engine processes the remaining frames within the chunk using the selected model. FiGO prunes the model ensemble to lower query optimization time. We empirically show that these techniques enable FiGO to outperform the state-of-the-art approaches for processing queries over videos by 3.3× on average across four video datasets.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback in improving the paper. This work was supported in part by NSF (CNS-1815047, IIS-1850342, and IIS-1908984), Cisco, Adobe, and Alibaba. This work was partially supported by Institute of Information and Communications Technology Planning and Evaluation grant funded by the Korea government (No. 2021-0-00766).

References

[1] Shipra Agrawal and Navin Goyal. 2012. Analysis of thompson sampling for the multi-armed bandit problem. In *COLT*.

[2] Michael R. Anderson, Michael Cafarella, German Ros, and Thomas F. Wenisch. 2019. Physical Representation-Based Predicate Optimization for a Visual Analytics Database. In *ICDE*. 1466–1477.

[3] Favyen Bastani, Songtao He, Arjun Balasingam, Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan, Michael Cafarella, Tim Kraska, and Sam Madden. 2020. MIRIS: Fast Object Track Queries in Video. In *SIGMOD*. 1907–1921.

[4] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G Andersen, Michael Kaminsky, and Subramanya R Dulloor. 2019. Scaling Video Analytics on Constrained Edge Nodes. In *SysML*.

[5] Olivier Chapelle and Lihong Li. 2011. An Empirical Evaluation of Thompson Sampling. In *NeurIPS*.

[6] Jack Chen, Samir Jindel, Robert Walzer, Rajkumar Sen, Nika Jimsheleishvilli, and Michael Andrews. 2016. The MemSQL Query Optimizer: A Modern Optimizer for Real-Time Analytics in a Distributed Database. In *PVLDB*. 1401–1412.

[7] Anil K Gupta, Ken G Smith, and Christina E Shalley. 2006. The interplay between exploration and exploitation. *Academy of management journal* (2006), 693–706.

[8] Brandon Haynes, Maureen Daum, Dong He, Amrita Mazumdar, Magdalena Balazinska, Alvin Cheung, and Luis Ceze. 2021. VSS: A Storage System for Video Analytics. In *SIGMOD*. 685–696.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. In *CVPR*. 770–778.

[10] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. 2018. Focus: Querying Large Video Datasets with Low Latency and Low Cost. In *OSDI*. 269–286.

[11] J. L. W. V. Jensen. 1906. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica* (1906), 175–193.

[12] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodík, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: scalable adaptation of video analytics. In *SIGCOMM*. 253–266.

[13] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics. In *PVLDB*. 533–546.

[14] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Deep CNN-Based Queries over Video Streams at Scale. In *PVLDB*. 1586–1597.

[15] Michael N Katehakis and Arthur F Veinott Jr. 1987. The multi-armed bandit problem: decomposition and computation. *Mathematics of Operations Research* (1987), 262–268.

[16] Sanjay Krishnan, Adam Dziedzic, and Aaron J Elmore. 2019. DeepLens: Towards a Visual Data Management System.

[17] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. In *ECCV*. 740–755.

[18] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single Shot MultiBox Detector. In *ECCV*. 21–37.

[19] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. 2018. Accelerating Machine Learning Inference with Probabilistic Predicates. In *SIGMOD*. 1493–1508.

[20] James G March. 1991. Exploration and exploitation in organizational learning. *Organization science* (1991), 71–87.

[21] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: a learned query optimizer. In *PVLDB*. 1705–1718.

[22] Oscar Moll, Favyen Bastani, Sam Madden, Mike Stonebraker, Vijay Gadepally, and Tim Kraska. 2020. ExSample: Efficient Searches on Video Repositories through Adaptive Sampling. *arXiv:2005.09141 [cs]* (2020).

[23] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, J. K. Aggarwal, Hyungtae Lee, Larry Davis, Eran Swears, Xioyang Wang, Qiang Ji, Kishore Reddy, Mubarak Shah, Carl Vondrick, Hamed Pirsiavash, Deva Ramanan, Jenny Yuen, Antonio Torralba, Bi Song, Anesco Fong, Amit Roy-Chowdhury, and Mita Desai. 2011. A Large-Scale Benchmark Dataset for Event Recognition in Surveillance Video. In *CVPR*. 3153–3160.

[24] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NeurIPS*.

[25] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NeurIPS*. 91–99.

[26] S.N.Bernstein. 1924. On a modification of Chebyshev's inequality and of the error formula of Laplace. *Ann. Sci. Inst. Sav. Ukraine, Sect. Math* (1924).

[27] Michael Stillger, Guy Lohman, Volker Markl, and Mokhtar Kandil. 2001. LEO – DB2's LEarning Optimizer. In *PVLDB*. 19–28.

[28] Mingxing Tan, Ruoming Pang, and Quoc V. Le. 2020. EfficientDet: Scalable and Efficient Object Detection. In *CVPR*. 10778–10787.

[29] William R. Thompson. 1933. On the Likelihood That One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika* (1933), 285–294.

[30] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. 2021. Scaled-YOLOv4: Scaling Cross Stage Partial Network. In *CVPR*. 13029–13038.

[31] Longyin Wen, Dawei Du, Zhaowei Cai, Zhen Lei, Ming-Ching Chang, Honggang Qi, Jongwoo Lim, Ming-Hsuan Yang, and Siwei Lyu. 2020. UA-DETRAC: A New Benchmark and Protocol for Multi-Object Detection and Tracking. In *CVIU*.

[32] Ioannis Xarchakos and Nick Koudas. 2019. SVQ: Streaming Video Queries. In *SIGMOD*. 2013–2016.

[33] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. 2020. BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning. (2020).

[34] Yuhao Zhang and Arun Kumar. 2019. Panorama: A Data System for Unbounded Vocabulary Querying over Video. In *PVLDB*. 477–491.