EVA: An End-to-End Exploratory Video Analytics System

Gaurav Tarlok Kakkar, Zhuangdi Xu, Prashanth Dintyala, Pramod Chunduri, Anirudh Prabakaran, Suryatej Reddy Vyalla, Jiashen Cao, Jaeho Bang, Abdullah Shah, Sanjana Garg, Aubhro Sengupta, Subrata Mitra[†], Ali Payani[‡], Yao Lu[⋆], Umakishore Ramachandran, Joy Arulraj
Georgia Institute of Technology [†]Adobe, [‡]Cisco, [⋆]Microsoft
arulraj@gatech.edu

Abstract

Computer vision has gone through a seismic shift over the last decade, with the availability of deep learning models capable of tackling diverse tasks like object detection and action localization. However, these models are highly computationally expensive, and leveraging them for analyzing videos involves low-level imperative programming. There is a growing interest in the database community in designing video database management systems (VDBMSs) to tackle these efficiency and usability challenges. Recently proposed VDBMSs, like PP (Microsoft) and BlazeIt (Stanford), accelerate declarative SQL-like queries over videos using database-inspired techniques like filtering irrelevant frames and modularizing vision pipelines. But, these systems have two key limitations. First, their optimizers are not tailored for offline, exploratory video analytics, thereby raising the cost of query processing. Second, they support a limited range of queries related to detecting objects and are unable to support richer queries like localizing actions. To tackle these issues, we present our vision of a VDBMS, called EVA, that optimizes exploratory queries using derived data structures and models by leveraging the user's accuracy requirement. EVA supports challenging vision queries using a set of novel adaptive query processing techniques that improve resource efficiency. This vision paper presents the architecture and the design decisions of EVA, our recent research questions and initial results.

1 Introduction

Advances in computer vision [10, 30] over the last decade has led to high interest among domain scientists and industry practitioners in leveraging vision models in their applications. However, there are efficiency and usability challenges associated with deploying vision pipelines in practice [18]. First, from a *resource efficiency* standpoint, these deep learning models are highly expensive to run on every frame of the video due to their depth (*i.e.*, number of neural network layers). Second, from a *usability standpoint*, the domain scientist must do low-level imperative programming across many libraries (*e.g.*, PyTorch [27], OpenCV [5], and Pandas [26]) to leverage these vision models.

To tackle these efficiency and usability challenges, database researchers have proposed VDBMSs [18, 24]. These systems improve usability by supporting declarative SQL-like queries over videos. For example, a movie analyst may issue the following query to study the distribution of gender of actors in a movie dataset [23]:

```
/* Movie Analysis */
SELECT GenderClassification(Crop(data, bbox))
FROM MOVIE CROSS APPLY
  UNNEST(FaceDetection(data)) AS Face(bbox, conf)
WHERE id > 1000 AND conf > 0.8;
```

Here, the query invokes *user-defined functions* (UDFs) that wrap around vision models [28]. It first retrieves the bounding boxes of all the faces present in each frame of the MOVIE video using the FaceDetection UDF [32] and the CROSS APPLY operator [33]. It filters out the faces for which the FaceDetection model has lower confidence (< 0.8). Next, it identifies the gender of each confidently-detected face using the GenderClassification UDF [22].

Prior Work. To efficiently process such queries, the state-of-the-art (SoTA) VDBMSs use a suite of database-inspired optimizations. For instance, PP trains a lightweight model to quickly filter out irrelevant frames (*e.g.*, frames that are not likely to contain a person), and only runs the heavyweight models on a subset of frames that pass through the filter model [24]. It reduces the query processing time and improves resource efficiency by reducing the *number of invocations* of the *heavyweight oracle models*.

What do Existing Systems Lack?. Through our collaboration with domain scientists at Georgia Tech on the eCloud project [4], we understood that these VDBMSs have two key limitations. First, their query optimizers are not tailored for *exploratory video analytics*, where the user iteratively refines the query to locate the target object [13]. They primarily focus on optimizing each query in isolation, even though these queries have significant *overlapping computation* (*e.g.*, redundant inference using a vision model over the same frame across queries). In our recent work [33], we showed that materializing and reusing the inference results of expensive models accelerates such exploratory workloads by 4× with negligible storage overhead for keeping the results (1.001×).

Second, these VDBMSs primarily focus on queries over objects detected in videos and do not support richer vision queries like localizing *actions*. An action refers to an event spread across a sequence of frames. For example, a movie analyst may be interested in querying specific actions (*e.g.*, a person dunking a basketball) in a movie dataset. It is challenging to accurately identify the basketball dunk action by running an object detector on each frame in isolation, as that requires context across frames [8]. These limitations significantly constrain the adoption of VDBMSs in practical applications.

Our Vision. To tackle these limitations, we are designing a novel VDBMS that is tailored for exploratory video analytics: EVA. It contains a sophisticated Cascades-style query optimizer for leveraging different forms of derived models and data structures. Similar to relational DBMSs, the Optimizer in EVA estimates the cost of query plan by profiling the cost of operators and estimating the selectivity of predicates. However, there is a key difference in that it must also optimize for query accuracy.

EVA seeks to support more complex vision tasks like action localization that are needed in practice. In our prior work [8], we

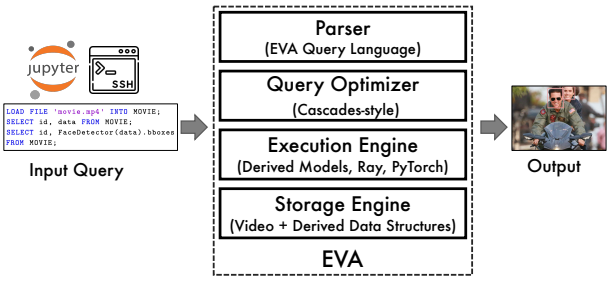

Figure 1: EVA System Architecture Overview

showed how to accomplish this by using an action classifier in a sliding window fashion over the video. The optimization problem lies in adaptively processing the input video at varying processing speeds. We are integrating an online feedback mechanism into the Optimizer of EVA, drawing inspiration from the adaptive query processing literature [11].

2 Design Philosophy

The design decisions in EVA are based on the following objectives:

- **Extensibility.** While EVA comes with a set of pre-loaded UDFs for frequently-used vision models (*e.g.*, FaceDetection, ObjectDetection), it allows users to define bespoke UDFs based on their requirements. The UDF may import additional Python packages and run arbitrary execution logic.
- **Modularity.** The user may compose UDFs with existing UDFs and other operators in the core VDBMS to construct complex queries. For example, the faces extracted using the FaceDetection UDF can be sent to the EmotionClassification model. This enables EVA to support new capabilities over time without losing backward compatibility.
- **Minimal Tuning.** We seek to automate query optimization as much as possible. The user need not manually remember to update statistics, or figure out how to partition the dataset. For example, EVA automatically materializes the results of expensive UDFs and reuses them to accelerate subsequent queries [33].
- **Inclusive.** EVA supports multiple DL frameworks and execution environments in an inclusive manner so that the user may choose whatever is demanded by their application. For example, the user may write UDFs in either PyTorch or TensorFlow based on the availability of off-the-shelf models for their target vision task.

3 Architecture of EVA

The architecture of the EVA VDBMS is shown in Fig. 1. We first present the query language that the Parser supports. We then describe the internals of the other three components.

3.1 EVA Query Language (EVAQL)

EVA's parser supports a query language tailored for exploratory video analytics, called EVAQL. The queries in this section all concern a movie dataset. EVA stores all the videos of this dataset in the following table:

```
MOVIE_DATA(
    ID SERIAL INTEGER,
    DATA NDARRAY UINT8(3, ANYDIM, ANYDIM),
    VIDEO_ID INTEGER,
    VIDEO_FRAME_ID INTEGER,
    VIDEO_NAME TEXT(30));
```

Listing 1: Schema of the movie dataset

Loading Data. EVA supports loading both videos and semi-structured data. The following query depicts how the user loads a video into the database:

```
/* Loading a video into the table */
LOAD FILE 'topgun.mp4' INTO MOVIE_DATASET;
```

To upload a video, the user need not explicitly create a table. EVA automatically creates a table called MOVIE_DATA with following columns: (1) id, (2) data, (3) video_id, (4) video_frame_id, and (5) video_name. They denote the frame identifier, the contents of the frame, and the video to which that frame belongs to.
EVAQL supports queries for loading structured data (*e.g.*, CSVs) for populating the metadata of videos (*e.g.*, bounding boxes of faces in a frame). Similar to traditional DBMSs, the user must explicitly define the schema before loading the CSV file:

```
/* Defining the schema and loading a CSV file */
CREATE TABLE IF NOT EXISTS MOVIE_METADATA (
    ID SERIAL INTEGER,
    VIDEO_ID INTEGER,
    VIDEO_FRAME_ID INTEGER,
    VIDEO_NAME TEXT(30),
    FACE_BBOXES NDARRAY FLOAT32(4));

LOAD FILE 'movie.csv'
    INTO MOVIE_METADATA WITH FORMAT CSV;
```

Besides these commands, EVA comes with scripts for bulk loading videos in a dataset. EVA expects that the videos in a dataset are of the same resolution. However, these videos may vary in duration (*i.e.*, the number of frames).

User-Defined Functions. EVAQL is tailored for supporting user-defined functions (UDFs). UDFs allow users to extend the VDBMS to support the requirements of their applications. In EVA, UDFs are often wrappers around deep learning models. For example, a face detection UDF takes a frame as input and returns the bounding boxes of the faces detected in the frame as output. Internally, it wraps around the FaceDetection PyTorch model [32].
EVAQL supports arbitrary UDFs that take a variety of inputs (*e.g.*, video meta-data or raw frames *etc.*) and generate a variety of outputs (*e.g.*, labels, bounding boxes, video frames, *etc.*). The following command registers a FaceDetection UDF in EVA:

```
/* Registering a User-Defined Function */
CREATE UDF IF NOT EXISTS FaceDetector
INPUT(frame NDARRAY UINT8(3, ANYDIM, ANYDIM))
OUTPUT(bboxes NDARRAY FLOAT32(ANYDIM, 4), scores
    NDARRAY FLOAT32(ANYDIM))
LOGICAL_TYPE  FaceDetection
IMPL '/udfs/face_detector.py'
PROPERTIES=('ACCURACY'='HIGH');
```

In § 3.1, the command specifies the input(s) and output(s) of the UDF. The FaceDetection UDF consumes a video frame of arbitrary dimensions (*e.g.*, 640 × 480 or 1280 × 720), and produces bounding boxes and confidence scores of the detected faces. IMPL specifies the path to the Python file containing the implementation of the

UDF. Internally, EVA uses `importlib` for creating an importing UDF objects from the file [12]. `LOGICAL_TYPE` specifies the model type of the UDF (*e.g.*, FaceDetection or ObjectDetection). The user must specify the target accuracy in `PROPERTIES`. EVA uses these properties to accelerate queries. For example, if the overall query accuracy requirement is moderate (*e.g.*, 0.8× the oracle model), EVA uses faster (but less accurate) models of the same model type to accelerate the query. After registering the UDF, it can be executed on a video as shown in § 1.

Interfaces. EVA currently supports EVAQL queries from both a command line interface and Jupyter notebooks. We seek to support other query languages similar to Pandas and Spark in the future.

3.2 Query Optimizer

EVA's Optimizer is based on the Cascades query optimization framework [16]. It applies a series of rules for rewriting the query and then performs cost-based optimization to generate a *physical query plan*.

The Optimizer in a VDBMS differs from that in a relational DBMS in two ways. First, it must focus on minimizing query processing time while meeting the accuracy constraint (which often does not exist in a typical relational DBMS). Second, it is expensive to derive statistics from videos a priori in exploratory video analytics, as that involves running expensive deep learning models. So, while processing an ad-hoc query, the Optimizer runs vision models on a subset of frames to guide important optimization decisions (*e.g.*, whether the query plan will meet the accuracy constraint or how should the predicates invoking vision models be ordered [33]).

UDFs are often the most expensive operators in VDBMS queries. To accelerate such queries, EVA materializes the results of the expensive UDFs and reuses them while processing subsequent queries in exploratory video analytics [33]. Reusing results of UDFs in VDBMSs differs from the query plan matching algorithms in traditional DBMSs [2] that primarily focus on expensive join operators. In contrast, in VDBMSs, UDFs frequently occur in predicates and projection lists.

EVA's optimizer supports novel general-purpose rewrite rules that are not present in SoTA VDBMSs [19, 24]. For example, to identify reuse opportunities, the Optimizer uses an UDF-centric rewrite rule (Fig. 2 (a)) that extracts the UDF from the predicate/projection expression and rewrites it using the CROSS APPLY operator [14]. The resulting query plan makes it feasible to explore rules like: (1) materializing and reusing results of the UDFs [33], (2) adding derived models (Fig. 2 (b)) [18, 19], (3) UDF reordering (Fig. 2 (c)), (4) UDF de-duplication, and (5) introducing a video sampling operator before the UDF. Here, UDF de-duplication refers to avoiding redundant computation of a UDF that occurs multiple times in a single query. For example, if both the UDFs in the left hand side query tree in Fig. 2(c) are identical, we merge them into a single apply operator to avoid redundant UDF evaluation.

3.3 Execution Engine

The Execution Engine is responsible for evaluating the query plan generated by the Optimizer. While executing the plan, it leverages heterogeneous computational units (*e.g.*, CPUs and GPUs). EVA leverages DL frameworks like PyTorch [27] for model inference.

In an earlier prototype of EVA [33], the Execution Engine did not support distributed query execution. We have subsequently added support for distributed query execution using Ray [25]. We describe our ongoing work on integrating Ray in § 4.1.

3.4 Storage Engine

Lastly, the Storage Engine is responsible for managing the videos. In an earlier prototype of EVA [33], the Storage Engine organized the videos as a sequence of decoded frames, similar to SoTA VDBMSs [18]. However, this approach significantly increases the storage footprint of EVA on larger datasets. Further, we found that it does not significantly reduce query execution time (§ 6). We have subsequently redesigned the Storage Engine to manage videos in a compressed format. The Storage Engine manages structured data (*e.g.*, bounding boxes of faces) on disk using the Parquet format [1]. It uses Arrow [29] as an in-memory columnar format for data that is being read or written using on-disk Parquet files. We are investigating the utility of a disk-based cache for video data.

4 Ongoing System Implementation

We are implementing EVA as a Python package that supports the client-server architecture [15]. EVA's client is compliant with the DB-API 2.0 specification [21]. The system is open-sourced under an Apache License [15]. We next describe our ongoing work in implementing EVA.

4.1 Integrating Ray

Our primary objective in integrating Ray into EVA is to support distributed query execution. We seek to initially support intra-query parallelism [17]. Consider a query that involves running the FaceDetection on a movie video with 13 K frames using a server with two GPUs. With a single GPU, it takes 402 s to process the query. Using Ray, EVA automatically splits the video into two partitions and uses both GPUs for model inference, reducing the query processing time to 209 s. Besides data-level parallelism, EVA also supports parallel processing of complex query predicates. For example, to evaluate: "UDF1(a) < 10 AND UDF2(b) > 20", the VDBMS may either evaluate the two atomic predicates in parallel, or perform canonical predicate reordering and short-circuit the predicate evaluation.

Why Ray?. We use the Ray framework instead of other distributed execution frameworks like Spark [34] for two reasons. First, Ray is designed for scaling DL applications, and has better support for managing GPUs. This feature is crucial for query plans involving multiple DL-based UDFs (*e.g.*, FaceDetection and GenderClassification). Second, Ray's actor model provides more flexibility than the data-level parallelism available in Spark. This allows EVA to better optimize queries (*e.g.*, pipelining operators for intra-query parallelism).

Exchange Operator. The Optimizer uses the exchange operator [6] to encapsulate the degree of parallelism (dop) in the query plan. The exchange operator splits the plan into two stages and configures the parallelism of the lower stage. Consider the query plan shown in Fig. 3. First, as specified by the lower exchange operator, two processes will run the FaceDetection UDF on the video. Then, the upper exchange operator indicates that a single process should run the GenderClassification UDF on the bounding boxes of the detected faces. To integrate Ray, we made the following modifications in EVA:

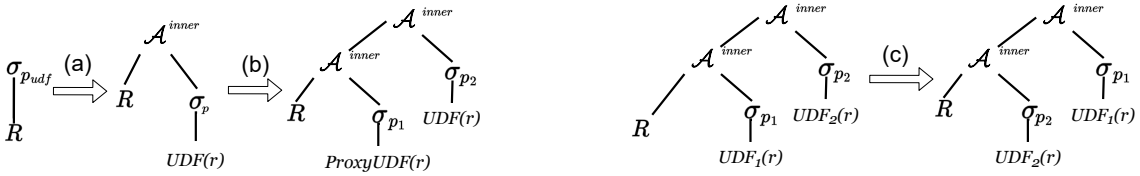

Figure 2: Illustrative UDF Optimization Rules – (a) UDF transformation rule that extracts the UDF from the predicate and converts to an `APPLY` operator, (b) UDF filtering rule that introduces a proxy UDF model for quickly filtering out irrelevant frames before executing UDF, and (c) UDF reordering rule that reorders UDFs based on their inference cost and availability of materialized results from prior queries.

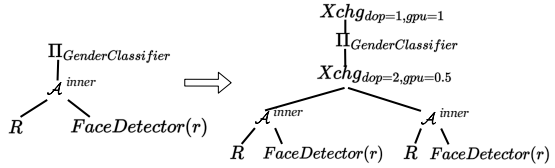

Figure 3: Illustration of Exchange Operator — This query retrieves the gender of all the faces detected in the video.

- **Execution Engine.** We convert the plan into Ray actors and chain them via Ray queues. For example, in Fig. 3, the CROSS APPLY operator and its children are converted into two Ray actors (*i.e.*, replicas), a_1 and a_2, that are each assigned half a GPU. Meanwhile, the PROJECTION operator is converted into single Ray actor, a_3, that is assigned one GPU. The output (*i.e.*, faces detected in the movie) of a_1 and a_2 is written into a Ray queue from which a_3 reads.
- **Optimizer.** Given the number of GPU accelerators and their computational capabilities, the Optimizer must decide where to inject the EXCHANGE operators in the query plan, and what is the suitable degree of parallelism for each operator. For example, in Fig. 3, the Optimizer injects an EXCHANGE operator above the CROSS APPLY operator (*i.e.*, FaceDetection) with *dop=2* and *gpu=0.5*, and another above the PROJECTION operator (*i.e.*, GenderClassification) with *dop=1* and *gpu=1*. We plan to extend the Optimizer to consider the cost of each UDF and effectively assign the resources to balance the pipeline, thereby optimizing the overall processing time.

4.2 Adaptive Query Processing

SoTA VDBMSs primarily focus on object-based queries over videos. These queries only require frame-level processing of videos and do not require the characterization of temporal information flow. As a result, such queries can be processed efficiently using optimizations such as batch processing and proxy models [18, 24]. However, for complex vision tasks such as action localization, even SoTA models operate at sub-optimal accuracy while requiring a large number of model parameters. So, batch processing and proxy models do not work well for action queries due to the large input/model sizes and the complexity of the task, respectively.

To accelerate such queries, EVA *adaptively* changes the input video segment while sequentially consuming the video. Specifically, Execution Engine continuously provides feedback (*e.g.*, model predictions) to the Optimizer. Based on the feedback, the Optimizer picks the subsequent input segment and sends it back to the Execution Engine. EVA leverages the SAMPLE operator to specify a set of sampling rates. EVA uses a greedy heuristic to select the sampling rate at each feedback step. It uses a lower sampling rate (more accurate) when the model prediction is `ACTION`, and higher sampling rates (less accurate) otherwise. We plan to support more parameters for picking the input video segment in the future (*e.g.*, resolution, stride) to further accelerate action queries.

5 Ongoing Research Efforts

5.1 Accuracy-Guided Query Optimization

As in relational DBMSs, the VDBMS's Optimizer estimates the query plan's *cost* by profiling the cost of the operators and estimating the selectivity of predicates. However, there are two key differences. First, DL models are *not* always accurate. So, unlike relational DBMSs, VDBMSs cannot guarantee accurate results. This gives the Optimizer an opportunity to jointly optimize the query plan for both runtime performance and accuracy requirements.

Second, the Optimizer must not treat an UDF as a black box. Instead, it should exploit the *semantic* properties of UDFs. For example, the Optimizer in EVA has the flexibility to pick a suitable *physical* model for processing a *logical* vision task, as long as it meets the query's accuracy constraint. In our prior work, we showed how the Optimizer may dynamically pick different models for processing video chunks of varying complexity [7]. We are investigating how to extend the Cascades-style Optimizer in EVA to jointly optimize for query execution cost and query accuracy. We seek to support complex model pipelines – proxy models, model cascades, and model ensembles.

5.2 Enhancing Query Capabilities

Our prior work in Zeus illustrated the importance of enhancing the query capability of VDBMSs to cover action queries [8]. However, Zeus has a fundamental limitation. It assumes the availability of a vision model explicitly trained for the target action (*e.g.*, a person riding a motorcycle). This assumption may *not* be valid in real-world applications for two reasons. First, the action may rarely occur in the dataset, leading to insufficient true positive examples (*i.e.*, class imbalance) during training. Second, the number of ad-hoc combinations of objects and their interactions that form the actions is exponential. Moreover, the action may span across an arbitrarily-long time interval. So, it is impractical to train a model that handles all these scenarios. To overcome these challenges, we are investigating techniques to break *ad-hoc actions* into a collection of *spatio-temporal predicates* over the *bounding boxes* and the *trajectories* of objects across a sequence of frames [9, 31].

To further enhance the query capabilities of VDBMSs, we seek to support re-identification and search queries in EVA. For example, consider a query that retrieves all the frames in a movie that contain a target actor. Efficiently searching for the specific actor using a

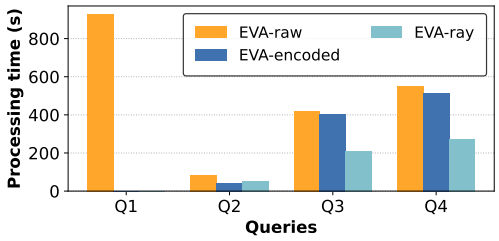

Figure 4: End-to-End Comparison — Comparison of different EVA variants on the four queries shown in Listing 2.

target image requires the use of computationally expensive object re-identification models. We are currently investigating the integration of incremental search techniques into EVA's Optimizer to accelerate re-identification queries.

5.3 GPU Optimization for EVA

Resource utilization. EVA supports UDFs with different performance characteristics (*e.g.*, some UDFs are more compute intensive, but others require more memory). Simply choosing a hard-coded execution parameter (*e.g.*, batch size) will not fully utilize available GPU resources and result in performance degradation during query execution. We are investigating how to adaptively choose these critical parameters to fully leverage GPUs.

Minimize data transfer cost. In queries with multiple UDFs, the same input frames may be transferred to the GPU multiple times (from the CPU) during query execution. For example, for query Q4 in our evaluation (§ 6), we observe that there is 10 GB additional data movement between the CPU and GPU. Second, EVA only has CPU implementations of certain operators like join, predicate filtering, and cropping. That results in data transfer between CPU and GPU between different operators. To minimize this cost, we seek to investigate two optimizations: (1) lazy eviction and (2) operator fusion. First, with lazy eviction, the Execution Engine caches the frames on GPU if they are required by later operators in the query pipeline. Second, with operator fusion, we plan to add GPU-centric implementations of general-purpose operators (*e.g.*, join and image cropping) to reduce data movement overhead.

6 End to End Evaluation

Hardware Setup. We perform the experiments on a server with the following specifications: 28 Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz, 2 NVIDIA Quadro P6000 GPUs, and 256 GB RAM.

Baselines. We construct three baselines using different variants of the EVA. EVA-raw is the first prototype of EVA, where it stores the video as decoded frames on disk [33]. EVA-encoded is the next version in which videos are stored as compressed files on disk and decoded on demand. EVA-ray improves upon EVA-encoded by integrating the Ray distributed execution engine.

Videos. We evaluate EVA on three movie video trailers: Bourne, Top Gun, and Hustle. Bourne clip consists of 13k frames, Top Gun consists of 3.6k frames, and Hustle consists of 3.8k frames. Bourne has a resolution of 1920 × 1080, while Top Gun and Hustle have a resolution of 1280 × 720. For examining action queries, we focus on the *Motorcycling* action in Top Gun (0.26 selectivity) and the *Basketball Dunk* action in Hustle (0.45 selectivity).

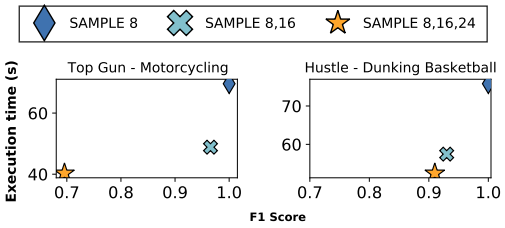

Figure 5: Action Queries in EVA — Execution time and accuracy (F1 score) for different sets of available sampling rates for the query shown in Listing 3 on two videos.

End-to-End Comparison. Fig. 4 compares the runtime performance of the EVA variants on the four queries listed in Listing 2.

```
Q1 LOAD FILE 'bourne.mp4' INTO MOVIE;
Q2 SELECT id, data FROM MOVIE;
Q3 SELECT id, FaceDetection(data).bboxes FROM MOVIE;
Q4 SELECT GenderClassification(Crop(data, bbox))
   FROM MOVIE  CROSS APPLY
    UNNEST(FaceDetection(data)) AS Face(bbox, conf)
   WHERE id > 1000 AND conf > 0.8;
```

Listing 2: Evaluation Queries – Q1 loads the video into the table `MOVIE`. Q2 scans through the video table. Q3 and Q4 analyse the MOVIE and uses FaceDetection and GenderClassification UDFs to extract faces and classify gender, respectively.

*Q*1 compares the video loading time. Since EVA-raw decodes the video before storing it on disk, it incurs a significant load overhead (924 s) compared to other baselines that leave the video compressed (0.1 s). *Q*2 compares the *video scan cost* across the baselines. There is no significant difference in performance. EVA-raw is slightly slower than the other baselines, as reading decoded frames has higher I/O cost than decoding frames on demand for the high-resolution video. EVA-ray suffers from the Ray's runtime initialization overhead and extra data movement overhead, and parallelism does not benefit much as there are no expensive computations. *Q*3 and *Q*4 compare the performance of executing queries with one or more UDFs. EVA-ray outperforms both EVA-raw and EVA-encoded by 1.9×. This is because it constructs a query plan to leverage both GPUs, as explained in § 4.1.

Action Localization. We next evaluate the performance of the adaptive query processing optimization discussed in § 4.2.

```
Q5 SELECT first(id) AS segment_id, ActionRecognizer(
     SEGMENT(data)) FROM MOVIE SAMPLE 8,16,24 GROUP
     BY '8f';
```

Listing 3: Action Query – Q5 applies an action recognition UDF.

Specifically, we measure the performance of EVA-encoded in executing query *Q*5 shown in Listing 3. `SEGMENT` constructs a video segment out of the frames, and `GROUP BY` determines the number of frames in a segment (*e.g.*, 8 frames). `SAMPLE 8` indicates that all segments are constructed using the single sampling rate 8 (*i.e.*, one frame out of every eight frames). So, EVA constructs a segment of 8 frames from 64 frames when the sampling rate is 8. `SAMPLE 8,16` adaptively varies the sampling rate between 8 and 16, and so on. We consider three different sets of available sampling rates in this experiment. Fig. 5 presents the execution time and accuracy (F1-score) of EVA on two videos with different action selectivities.

The most notable observation is that the availability of a higher sampling rates allows EVA-ENCODED to lower the execution time on both videos. The speedup achieved on Top Gun is more than that on Hustle, since the action selectivity is lower in Top Gun, thereby allowing EVA to more frequently use a higher sampling rate. However, the lower action selectivity also results in a more drastic drop in accuracy in Top Gun (0.3 F1 points).

7 Related Work

We now discuss how EVA is related to two SoTA VDBMSs.

VIVA. [20] is an end-to-end VDBMS for interactive video analytics. It allows users to provide domain knowledge using *hints* that the OPTIMIZER leverages for optimizing queries. We plan to extend EVA's OPTIMIZER to support such hints as they may further improve the quality of plans. Similar to VIVA, EVA supports joint optimization of structured and unstructured data. These optimizations are described in [33].

VOCAL. [9] is a SoTA VDBMS that supports interactive compositional queries. The key idea is to leverage user feedback via active learning to better explore the video. We plan to adopt this approach in EVA to further enhance its query capabilities (§ 5.2). In particular, the user may break down action queries into a collection of spatio-temporal predicates over the bounding boxes and trajectories of objects across a sequence of frames. We anticipate that the compositional queries enabled in VOCAL will be further accelerated by the optimizations in EVA.

We next provide an overview of other SoTA VDBMSs.

Derived Models and Data Structures. SoTA VDBMSs construct models or data structures *derived* from the original video to reduce the number of invocations of the heavyweight, oracle model. PP [19, 24] filters out irrelevant frames using a lightweight proxy model. BLAZEIT [18] constructs a bespoke aggregation model to directly answer the given aggregate query. TAHOMA [3] uses a cascade of scaled models of increasing complexity and short-circuits the inference when the required accuracy constraint is met. FIGO [7] uses an ensemble of off-the-shelf scaled models of increasing complexity. It does not train any derived models to process ad-hoc queries, enabling it to work across diverse queries and video datasets seamlessly. EVA's OPTIMIZER seeks to leverage these derived models and data structures in a holistic manner.

8 Conclusion

We presented EVA, our vision for an end-to-end exploratory video analytics system. We discussed how EVA accelerates vision queries by materializing and reusing the results of expensive UDFs, by supporting data- and query-level parallelism, and through holistic query optimization. EVA seeks to expand the querying capabilities of VDBMSs to facilitate their adoption in practical applications. We hope that EVA allows a broader set of application developers to benefit from recent advances in vision and database systems.

References

[1] Apache software foundation. 2013. Apache Parquet. https://parquet.apache.org/.

[2] S. R. Alekh Jindal, Konstantions Karanasos and H. Patel. Selecting Subexpressions to Materialize at Datacenter Scale. In *VLDB*, 2018.

[3] M. R. Anderson, M. J. Cafarella, G. Ros, and T. F. Wenisch. Physical representation-based predicate optimization for a visual analytics database. *CoRR*, abs/1806.04226, 2018.

[4] J. Arulraj, A. Chatterjee, A. Daglis, A. Dhekne, and U. Ramachandran. eCloud: A Vision for the Evolution of the Edge-Cloud Continuum. *Computer*, 54(5):24–33, 2021. Publisher: IEEE.

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

[6] E. Brewer. Volcano & the Exchange Operator, 2022.

[7] J. Cao, K. Sarkar, R. Hadidi, J. Arulraj, and H. Kim. FiGO: Fine-Grained Query Optimization in Video Analytics. In *SIGMOD*, pages 559–572, 2022. event-place: Philadelphia, PA, USA.

[8] P. Chunduri, J. Bang, Y. Lu, and J. Arulraj. Zeus: Efficiently Localizing Actions in Videos Using Reinforcement Learning. In *SIGMOD*, pages 545–558, 2022.

[9] M. Daum, E. Zhang, D. He, M. Balazinska, B. Haynes, R. Krishna, A. Craig, and A. Wirsing. VOCAL: Video Organization and Interactive Compositional AnaLytics. In *CIDR*, 2022.

[10] J. Dean, D. Patterson, and C. Young. A new golden age in computer architecture: Empowering the machine-learning revolution. *MICRO*, 38(2):21–29, 2018. Publisher: IEEE.

[11] A. Deshpande, Z. Ives, V. Raman, et al. Adaptive query processing. *Foundations and Trends® in Databases*, 1(1):1–140, 2007.

[12] P. S. Foundation. Importlib - the implementation of import, 2022.

[13] A. Galakatos, A. Crotty, E. Zgraggen, C. Binnig, and T. Kraska. Revisiting Reuse for Approximate Query Processing. *VLDB*, 10:1142–1153, 2017.

[14] C. Galindo-Legaria and M. Joshi. Orthogonal optimization of subqueries and aggregation. In *SIGMOD '01*, 2001.

[15] Georgia Tech Database Group. EVA Video Database System. https://pypi.org/project/evadb/, 2022.

[16] G. Graefe. The Cascades Framework for Query Optimization. *IEEE Data Eng. Bull.*, 18(3):19–29, 1995.

[17] N. Hardavellas and I. Pandis. *Intra-Query Parallelism*, pages 1567–1568. Springer US, Boston, MA, 2009.

[18] D. Kang, P. Bailis, and M. Zaharia. BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics. *Proc. VLDB Endow.*, 13:533–546, 2019.

[19] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia. NoScope: Optimizing Neural Network Queries over Video at Scale. *VLDB*, 10(11):1586–1597, Aug. 2017. Publisher: VLDB Endowment.

[20] D. Kang, F. Romero, P. D. Bailis, C. Kozyrakis, and M. Zaharia. VIVA: an end-to-end system for interactive video analytics. In *CIDR*, 2022.

[21] M.-A. Lemburg. Python database api specification v2.0. PEP 249, 2001.

[22] G. Levi and T. Hassncer. Age and gender classification using convolutional neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 34–42, 2015.

[23] G. Liu, H. Shi, A. Kiani, A. Khreishah, J. Lee, N. Ansari, C. Liu, and M. M. Yousef. Smart Traffic Monitoring System using Computer Vision and Edge Computing. *IEEE Transactions on Intelligent Transportation Systems*, 2021. Publisher: IEEE.

[24] Y. Lu, A. Chowdhery, S. Kandula, and S. Chaudhuri. Accelerating Machine Learning Inference with Probabilistic Predicates. *SIGMOD*, 2018.

[25] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 561–577, Carlsbad, CA, Oct. 2018. USENIX Association.

[26] T. pandas development team. pandas-dev/pandas: Pandas, Feb. 2020.

[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*, 2019.

[28] A. Rheinländer, U. Leser, and G. Graefe. Optimization of complex dataflows with user-defined functions. *ACM Computing Surveys (CSUR)*, 50(3):1–39, 2017. Publisher: ACM New York, NY, USA.

[29] N. Richardson, I. Cook, N. Crane, D. Dunnington, R. François, J. Keane, D. Moldovan-Grünfeld, J. Ooms, and Apache Arrow. *arrow: Integration to Apache Arrow*, 2022. https://github.com/apache/arrow/, https://arrow.apache.org/docs/r/.

[30] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, and M. Bernstein. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. Publisher: Springer.

[31] M. A. Sakr and R. H. Güting. Spatiotemporal pattern queries. *GeoInformatica*, 15(3):497–540, 2011.

[32] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823, 2015.

[33] Z. Xu, G. T. Kakkar, J. Arulraj, and U. Ramachandran. EVA: A Symbolic Approach to Accelerating Exploratory Video Analytics with Materialized Views. In *SIGMOD*, pages 602–616, 2022.

[34] M. A. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *HotCloud*, 2010.