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

Gaurav Tarlok Kakkar, Jiashen Cao, Pramod Chunduri, Zhuangdi Xu, Suryatej Reddy Vyalla, Prashanth Dintyala, Anirudh Prabakaran, Jaeho Bang, Aubhro Sengupta, Kaushik Ravichandran, Ishwarya Sivakumar, Aryan Rajoria, Ashmita Raju, Tushar Aggarwal, Abdullah Shah, Sanjana Garg, Shashank Suman, Myna Prasanna Kalluraya,
Subrata Mitra[†], Ali Payani[‡], Yao Lu[★], Umakishore Ramachandran, Joy Arulraj
Georgia Institute of Technology [†]Adobe, [‡]Cisco, [★]Microsoft
arulraj@gatech.edu

Abstract

In recent years, deep learning models have revolutionized computer vision, enabling diverse applications. However, these models are computationally expensive, and leveraging them for video analytics involves low-level imperative programming. To address these efficiency and usability challenges, the database community has developed video database management systems (VDBMSs). However, existing VDBMSs lack extensibility and composability and do not support holistic system optimizations, limiting their practical application. In response to these issues, we present our vision for EVA, a VDBMS that allows for extensible support of user-defined functions and employs a Cascades-style query optimizer. Additionally, we leverage Ray's distributed execution to enhance scalability and performance and explore hardware-specific optimizations to facilitate runtime optimizations. We discuss the architecture and design of EVA, our achievements thus far, and our research roadmap.

ACM Reference Format:
Gaurav Tarlok Kakkar, Jiashen Cao, Pramod Chunduri, Zhuangdi Xu, Suryatej Reddy Vyalla, Prashanth Dintyala, Anirudh Prabakaran, Jaeho Bang, Aubhro Sengupta, Kaushik Ravichandran, Ishwarya Sivakumar, Aryan Rajoria, Ashmita Raju, Tushar Aggarwal, Abdullah Shah, Sanjana Garg, Shashank Suman, Myna Prasanna Kalluraya,, Subrata Mitra[†], Ali Payani[‡], Yao Lu[★], Umakishore Ramachandran, Joy Arulraj. 2023. EVA: An End-to-End Exploratory Video Analytics System. In *Data Management for End-to-End Machine Learning (DEEM '23), June 18, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3595360.3595858

1 INTRODUCTION

Advances in computer vision [11, 32] 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 [20]. 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 [28], OpenCV [5], and Pandas [26]) to leverage these vision models. To tackle these efficiency and usability challenges, database researchers have proposed several video database management systems (VDBMSs) [4, 9, 20, 22, 24, 34]. These systems improve usability by supporting declarative SQL-like queries over videos.

VDBMSs have applications across several domains, including movie analysis, monitor wildlife behavior [12, 19], monitor traffic [41], analyze retail store performance [36]. For example, a movie analyst may issue the following query to study the emotion palette of actors in a movie dataset [23]:

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

Listing 1: Illustrative EVAQL query

Here, the query invokes *user-defined functions* (UDFs) that wrap around vision models [29]. It first retrieves the bounding boxes of all the faces present in the initial 1000 frames of the MOVIE video using the FaceDetection UDF [35]. It filters out the faces for which the FaceDetection model has lower confidence (< 0.8). Next, it identifies the emotion of each confidently-detected face using EmotionClassification UDF.

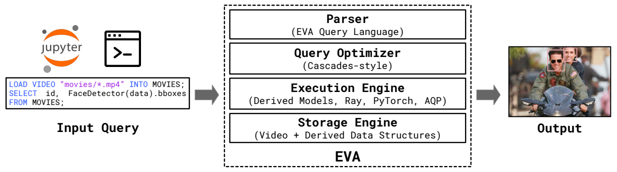
Prior Work. To efficiently process such queries, the state-of-the-art (SoTA) VDBMSs use a suite of database-inspired optimizations. For instance, PP [24] trains a lightweight model to quickly filter out irrelevant frames (*e.g.*, frames that are not likely to contain a person), and only runs the heavyweight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

❶ **Extensibility and Composability**: They do not allow users to define their own user-defined functions (UDFs) for vision models, and lack the ability of *compose* UDFs to accomplish complex tasks (Listing 1). Furthermore, these VDBMSs mainly focus on queries over detected video objects and do not support richer vision queries like *action localization* [37].

❷ **Holistic System Optimizations**: Prior systems primarily focus on optimizing each query in isolation, even though workloads have significant overlapping computation (*e.g.*, redundant inference using a vision model over the same frame across queries) [40]. They often use lightweight proxy models to accelerate query execution.

DEEM '23, June 18, 2023, Seattle, WA, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0204-4/23/06.
https://doi.org/10.1145/3595360.3595858



```
LOAD VIDEO "movies/*.mp4" INTO MOVIES;
SELECT  id,  FaceDetector(data).bboxes
FROM MOVIES;
```

Figure 1: Architecture of EVA

So, they do not support holistic optimization for more complex queries, both during query optimization and execution. These limitations significantly constrain the adoption of VDBMSs in practical applications. Raven [27] optimizes ML and relational pipelines with cross-query optimization. Gandhi *et al.* [16] utilizes tensor abstraction for trainable pipelines in AI and relational workloads. We plan to support the training pipeline in the future.

Our Vision. To overcome existing limitations, we're developing an innovative VDBMS that's specifically designed for exploratory video analytics - EVA. EVA provides extensible support for UDFs(§ 3.1), allowing users to define bespoke UDFs based on their requirements and compose them with existing UDFs and operators to construct complex queries. For example, the FaceDetection and EmotionClassification models can be used to construct an emotion detection query. Additionally, UDFs can import third-party Python packages and execute arbitrary logic, which makes it easy for EVA to support new features in the future.

To optimize query plans, EVA contains a Cascades-style query optimizer (§ 3.2) that leverages different forms of derived models and data structures. Like relational DBMSs, EVA estimates the cost of query plans by profiling operator costs and estimating predicate selectivity. It goes further by optimizing for query accuracy (§ 4.2). Moreover, EVA's distributed Execution Engine powered by Ray (§ 3.3) provides additional scalability and performance. We're also exploring hardware-specific optimizations and drawing inspiration from the adaptive query processing literature [13] to facilitate runtime optimizations (§ 4.3).

2 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.

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

Listing 2: Schema of the movie dataset

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

```
/* Loading a video into the table */
LOAD VIDEO 'movies/*.mp4' INTO MOVIE_DATA;
```

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 CSV 'movie.csv' INTO MOVIE_METADATA;
```

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 a FaceDetection PyTorch model [35].

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
TYPE  FaceDetection
IMPL '/udfs/face_detector.py'
PROPERTIES ('ACCURACY'='HIGH');
```

`TYPE` specifies the logical model type of the UDF (*e.g.*, FaceDetection or ObjectDetection). `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 [14]. The user can specify other metadata like the 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 Listing 1.

Interfaces. EVA currently supports EVAQL queries from both a command line interface and Jupyter notebooks. We seek to support a Pythonic dataframe API in the future.

2.2 Query Optimizer

EVA's Optimizer is based on the Cascades framework [17]. 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 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 [22, 31, 40]).

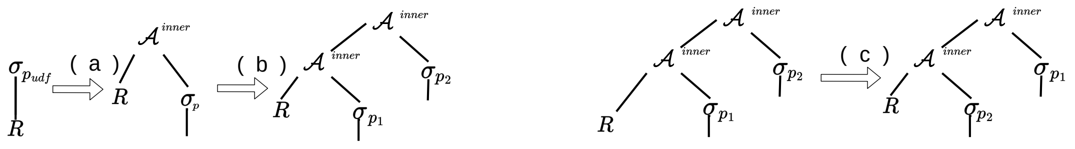


Figure 2: Illustrative UDF Optimization Rules – (a) UDF transformation rule that extracts the UDF from the predicate and converts to an APPLY operator, (b) UDF filtering rule that introduces a proxy UDF model for quickly filtering out irrelevant frames before executing UDF, and (c) UDF reordering rule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

2.3 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 [28] for model inference. In an earlier prototype of EVA [40], the Execution Engine did not support distributed query execution. We have recently added support for distributed query execution (§ 3.3) using Ray [25].

2.4 Storage Engine

Lastly, the Storage Engine is responsible for managing the videos. In an earlier prototype of EVA [40], the Storage Engine organized the videos as a sequence of decoded frames, similar to SoTA VDBMSs [20]. However, this approach not only significantly increases the storage footprint of EVA on larger datasets but also does not provide any significant reduction in query execution time. 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 [30] as an in-memory columnar format for data that is being read or written using on-disk Parquet files.

3 PROGRESS

We are implementing EVA as a Python package with Apache License [2] based on a client-server architecture [2]. We have made progress on enhancing the extensibility of EVA, and the efficacy of the Optimizer and the Execution Engine.

3.1 Extensibility - Importing UDFs

EVA allows users to import their own UDFs in two ways. Users can either import their own implemented UDFs (i.e., from source) or from popular third party platform (*e.g.*, HuggingFace [39], PyTorch).

UDF from Source.

```
# Configuring an UDF with decorators
class ImageClassificationUDF:
    @setup(cachable=True, batchable=True,
           udf_type="ImageClassification")
    def setup(self): # prepare the UDF
    @forward(
    input_signatures=[PyTorchTensor(
                      type=NdArrayType.FLOAT32,
                      dimensions=(1,3,540,960))],
    output_signatures=[PandasDataframe(
                       columns=["label"],
                       column_types=[NdArrayType.STR
                           ])]
    )
    def forward(self): # do inference
```

EVA supports defining UDFs using function *decorators* in Python. This allows users to migrate their existing deep learning models to EVA with a few lines of Python code. Users define the input and output formats of their models and configuration options through the decorator-based syntax. In Listing 3.1, the `@setup` decorator specifies the configuration options for the UDF. The user specifies the properties – whether EVA can cache results of UDF, does the UDF support batch mode execution, *etc.*. The `@forward` decorator specifies the input and output types/dimensions for the UDF.

UDF from HuggingFace. Recently, HuggingFace [39] has gained popularity amongst the deep learning community for their support of various models across multiple data modalities (*e.g.*, text, audio, video, *etc.*). EVA supports HuggingFace tasks and models right out of the box. Users define tasks or specify models using EVA's declarative language:

```
/* Registering an ObjectDetectorModel */
CREATE UDF MyObjectDetector TYPE HuggingFace
PROPERTIES ('task'='object-detection',
            'model'='facebook/detr-resnet-50')
```

Here, the user adds UDF that performs `object-detection` using the model `facebook/detr-resnet-50`.

3.2 Query Optimizer - Reuse of Inference Results

UDFs are often the most expensive operators in VDBMS queries. To accelerate such queries, EVA materializes the results of UDFs and reuses them while processing subsequent queries in exploratory video analytics [40]. Reusing results of UDFs in VDBMSs differs from the query plan matching algorithms in traditional DBMSs [3] that 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. 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 [15]. The resulting query plan makes it feasible to explore rules like: (1) materializing and reusing results of the UDFs [40], (2) adding derived models (Fig. 2 (b)) [20, 21], (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.

3.3 Execution Engine - Integrating Ray

Our primary objective in integrating Ray into EVA is to support distributed query execution. We seek to initially support intra-query parallelism [18]. Consider a query that involves running

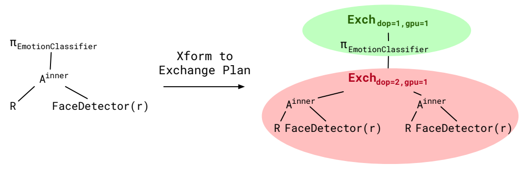


Figure 3: Illustration of Exchange Operator — This query retrieves the emotions of all the faces detected in the video.

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.

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 EmotionClassification UDF on the bounding boxes of the detected faces. To leverage Ray, the Optimizer in EVA transforms the query plan into Ray actors and chains them via Ray queues.

4 ROADMAP

We next describe our ongoing work and open questions in implementing EVA. We seek to continue improving the usability of EVA, and also the efficacy of the Optimizer and the Execution Engine.

4.1 Extensibility - Enhancing Querying Capability

Action Queries. In our prior work in Zeus [8], we emphasized the need to improve the querying capabilities of VDBMSs to encompass action queries. Zeus assumes the availability of a vision model explicitly trained for the target action (*e.g.*, a person riding a motorcycle). However, in real-world applications the action may rarely occur in the dataset, leading to insufficient true positive examples (*i.e.*, class imbalance) during training. In addition, the number of ad-hoc combinations of objects and their interactions that form the actions is exponential. To overcome these challenges, we seek to pursue a more practical approach in EVA. 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 [10, 33].

Similarity Search. To meet the needs of real-world applications [38], we seek to support object re-identification and similarity search queries in EVA. Consider a query that retrieves all the frames in a movie that contain a target actor. Efficiently searching for the specific actor using a 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.

4.2 Query Optimizer - Accuracy-Guided 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, deep learning 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 constraints.

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 [7], we showed how the Optimizer may dynamically pick different models for processing video chunks of varying complexity. 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.

4.3 Execution Engine - GPU-aware Optimization

Resource utilization. As EVA extensively uses GPUs for query processing, it is critical to optimize query execution on GPUs. The Optimizer needs to insert the exchange operator and tune the degree-of-parallelism (DOP) parameter. The optimal DOP value depends on the model execution cost, the overall query, and the underlying data. We are investigating how to optimize this critical parameter to better leverage GPUs. Concretely, given the number of GPUs and their computational capabilities, EVA must decide where to inject the exchange operators in the query plan, and what is the suitable degree of parallelism for each operator. To achieve this, the Optimizer first generates a statically optimized plan. Later, it leverages the adaptive Execution Engine by adjusting the pipeline dynamically during execution to reduce overall processing time.

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. 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 (*e.g.*, 10-GB additional data movement for the query shown in Listing 1. 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.

5 CONCLUSION

In this paper, we present our vision, current progress, and road map for future improvements on EVA, focusing on querying capability, query optimization, and query execution. We hope that EVA will enable a broader set of application developers to leverage recent advances in vision for analysing unstructured data.

References

[1] Apache Parquet. https://parquet.apache.org/.
[2] EVA Video Database System. https://pypi.org/project/evadb/.
[3] S. R. Alekh Jindal, Konstantions Karanasos and H. Patel. Selecting Subexpressions to Materialize at Datacenter Scale. In *VLDB*, 2018.
[4] F. Bastani, S. He, A. Balasingam, K. Gopalakrishnan, M. Alizadeh, H. Balakrishnan, M. Cafarella, T. Kraska, and S. Madden. MIRIS: Fast Object Track Queries in Video. In *SIGMOD*, pages 1907–1921, 2020.
[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, B. Haynes, D. He, A. Mazumdar, M. Balazinska, and A. Cheung. TASM: A Tile-Based Storage Manager for Video Analytics. *ArXiv*, abs/2006.02958, 2020.
[10] 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.
[11] 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.
[12] J. Dellinger, C. Shores, A. Craig, S. Kachel, M. Heithaus, W. Ripple, and A. Wirsing. Predators reduce niche overlap between sympatric prey. *Oikos*, 12 2021.
[13] A. Deshpande, Z. Ives, V. Raman, et al. Adaptive query processing. *Foundations and Trends in Databases*, 1(1):1–140, 2007.
[14] P. S. Foundation. Importlib - the implementation of import, 2022.
[15] C. Galindo-Legaria and M. Joshi. Orthogonal optimization of subqueries and aggregation. In *SIGMOD '01*, 2001.
[16] A. Gandhi, Y. Asada, V. Fu, A. Gemawat, L. Zhang, R. Sen, C. Curino, J. Camacho-Rodríguez, and M. Interlandi. The Tensor Data Platform: Towards an AI-centric Database System. *CIDR*, 2023.
[17] G. Graefe. The Cascades Framework for Query Optimization. *IEEE Data Eng. Bull.*, 18(3):19–29, 1995.
[18] N. Hardavellas and I. Pandis. *Intra-Query Parallelism*, pages 1567–1568. Springer US, Boston, MA, 2009.
[19] M. Heithaus, L. Dill, G. Marshall, and B. Buhleier. Habitat use and foraging behavior of tiger sharks (galeocerdo cuvier) in a seagrass ecosystem. *Marine Biology*, 140(2):237–248, 2002.
[20] 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.
[21] 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.
[22] 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.
[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] Pandas. pandas-dev/pandas: Pandas, Feb. 2020.
[27] K. Park, K. Saur, D. Banda, R. Sen, M. Interlandi, and K. Karanasos. End-to-end Optimization of Machine Learning Prediction Queries. In *SIGMOD*, pages 587–601, 2022.
[28] 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.
[29] 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.
[30] 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/.
[31] F. Romero, J. Hauswald, A. Partap, D. Kang, M. Zaharia, and C. Kozyrakis. Optimizing video analytics with declarative model relationships. *Proc. VLDB Endow.*, 16(3):447–460, 2022.
[32] 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.
[33] M. A. Sakr and R. H. Güting. Spatiotemporal pattern queries. *GeoInformatica*, 15(3):497–540, 2011.
[34] M. Satyanarayanan, P. B. Gibbons, L. B. Mummert, P. Pillai, P. Simoens, and R. Sukthankar. Cloudlet-based just-in-time indexing of iot video. In *Global Internet of Things Summit, GIoTS 2017, Geneva, Switzerland, June 6-9, 2017*, pages 1–8. IEEE, 2017.
[35] 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.
[36] A. W. Senior, L. M. Brown, A. Hampapur, C. Shu, Y. Zhai, R. S. Feris, Y. Tian, S. Borger, and C. R. Carlson. Video analytics for retail. In *AVSS*, pages 423–428. IEEE Computer Society, 2007.
[37] Z. Shou, D. Wang, and S.-F. Chang. Temporal action localization in untrimmed videos via multi-stage cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1049–1058, 2016.
[38] T. Skopal, F. Falchi, J. Lokoc, M. L. Sapino, I. Bartolini, and M. Patella, editors. *Similarity Search and Applications - 15th International Conference, SISAP 2022, Bologna, Italy, October 5-7, 2022, Proceedings*, volume 13590 of *Lecture Notes in Computer Science*. Springer, 2022.
[39] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics.
[40] 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.
[41] S. Yang, E. Bailey, Z. Yang, J. Ostrometzky, G. Zussman, I. Seskar, and Z. Kostic. COSMOS smart intersection: Edge compute and communications for bird's eye object tracking. In *PerCom*, pages 1–7. IEEE, 2020.