Interactive Demonstration of EVA

Gaurav Tarlok Kakkar
Georgia Institute of Technology
gkakkar7@gatech.edu

Aryan Rajoria
Bennett University
e19cse319@bennett.edu.in

Myna Prasanna Kalluraya
Georgia Institute of Technology
mkalluraya6@gatech.edu

Ashmita Raju
Georgia Institute of Technology
ashmita.raju@gatech.edu

Jiashen Cao
Georgia Institute of Technology
jiashenc@gatech.edu

Kexin Rong
Georgia Institute of Technology
krong@gatech.edu

Joy Arulraj
Georgia Institute of Technology
arulraj@gatech.edu

ABSTRACT

In this demonstration, we will present EVA, an end-to-end AI-Relational database management system. We will demonstrate the capabilities and utility of EVA using three usage scenarios: (1) EVA serves as a backend for an exploratory video analytics interface developed using Streamlit and React, (2) EVA seamlessly integrates with the Python and Data Science ecosystems by allowing users to access EVA in a Python notebook alongside other popular libraries such as Pandas and Matplotlib, and (3) EVA facilitates bulk labeling with Label Studio, a widely-used labeling framework. By optimizing complex vision queries, we illustrate how EVA allows a wide range of application developers to harness the recent advances in computer vision.

PVLDB Reference Format:
Gaurav Tarlok Kakkar, Aryan Rajoria, Myna Prasanna Kalluraya, Ashmita Raju, Jiashen Cao, Kexin Rong, and Joy Arulraj. Interactive Demonstration of EVA. PVLDB, 16(12): 4082 - 4085, 2023.
doi:10.14778/3611540.3611626

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/georgia-tech-db/evadb.

1 INTRODUCTION

Over the last decade, advances in computer vision [7, 17] have sparked significant interest among domain scientists and industry practitioners in integrating vision models into their applications. However, deploying vision pipelines in practice comes with efficiency and usability challenges [19], such as the high computational cost of running deep learning models, and the need for low-level imperative programming across multiple libraries (*e.g.*, Pandas [14], OpenCV [3], PyTorch [15]). To address these challenges, researchers have proposed a wide range of video database management systems (VDBMSs) [2, 6, 11, 12] that support declarative SQL-like queries over videos.

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611626

```
/* Movie Emotion Analysis */
SELECT id, EmotionClassification(Crop(data, bbox))
FROM MOVIE SAMPLE "5s" CROSS APPLY
    UNNEST(FaceDetection(data)) AS Face(bbox, conf)
WHERE id > 1000 AND conf > 0.8
ORDER BY Similarity(Feature("tom_cruise.png"),
    Feature(Crop(data, Face.bbox))
LIMIT 10;
```

Listing 1: Illustrative EVAQL query

Limitations of Existing Systems. These systems have two key limitations:

1. **Usability.** First, they have limited support for user-defined functions (UDFs). Users cannot easily define custom UDFs that wrap around deep learning models. Users also cannot compose multiple UDFs in a single query to accomplish complex tasks. Consider the query shown in Listing 1, where the user seeks to examine the emotions of Tom Cruise in the latest Top Gun movie. To achieve this, the user first applies a `FaceDetection` UDF to extract the face from the video and then uses a `Feature` UDF to measure similarity of the detected face against an image of Tom Cruise. Finally, an EmotionClassification UDF is used to classify the emotions. The user can also specify the sampling rate of the video to apply the UDFs every 5 seconds. We seek to support such complex queries in EVA [10].
2. **Efficiency.** Another limitation of existing VDBMSs is that they primarily focus on optimizing individual queries in isolation, which leads to missed optimization opportunities across exploratory queries [19].

Our Approach. EVA [10] seeks to address these limitations by simplifying the process of adding UDFs, supporting queries that invoke multiple UDFs, and optimizing exploratory query workloads.

EVA allows users to define bespoke UDFs based on their requirements, and compose them with existing UDFs and operators to construct complex queries (§ 2.1). For example, the FaceDetection and EmotionClassification models can be used to construct an emotion detection query. Users may easily import third-party Python packages in UDFs to support complex logic. This improves the extensibility and usability of EVA.

To improve the efficiency of query execution, EVA uses a Cascades-style query optimizer (§ 2.2) that jointly optimizes for accuracy and

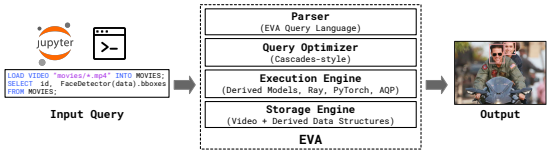

Figure 1: Architecture of EVA

cost. EVA also supports a distributed Execution Engine across multiple GPUs, powered by Ray, to reduce query processing time [10].

Usage Scenarios. To demonstrate the capabilities and utility of EVA, we focus on three different scenarios in this demo:

1. **EVA-UI.** In the first scenario, we present EVA-UI, an easy-to-use visual interface for performing exploratory video analytics. EVA-UI allows users to explore visual data and gain insights through an intuitive user interface. The EVA-UI frontend is backed by the EVA backend.
2. **Integration with Data Science Ecosystem.** In the second scenario, we illustrate how EVA seamlessly integrates with the Python and Data Science (DS) ecosystems. We will demonstrate how the Python APIs provided by EVA allow users to access EVA in a Python notebook and use it alongside other popular libraries such as Pandas and Matplotlib. This workflow empowers users to perform video analytics efficiently and with ease.
3. **Integration with LabelStudio.** Lastly, we discuss how EVA complements Label Studio, a widely-used labeling framework. Through this integration, EVA reduces labeling time by allowing users to label multiple frames at a time (*i.e.*, bulk labeling). EVA supports a similarity search UDF that allows users to propagate one label to multiple similar images, therefore reducing the overall labeling time. We note that we make no modifications in the Label Studio framework. This scenario highlights how existing Python libraries may leverage a VDBMS to take advantage of recent advances in the vision for analyzing unstructured data.

In all of the scenarios, we will let the attendees modify queries, UDFs, and observe the performance impact of using EVA.

2 SYSTEM OVERVIEW

In this section, we present an overview of the EVA VDBMS (Fig. 1). EVA has four components : parser, optimizer, execution and storage engine. The key design choice of EVA is to provide a SQL interface to users, with a client-server architecture compliant with the DB-API 2.0 specification [13]. The SQL interface is already widely used by many other database systems and is easy to extend. It plays a key role to allow users to integrate EVA with their applications in different scenarios. We provide more details in § 2.1 about the SQL interface. We also briefly describe the internals of the other three components of the system in § 2.2.

2.1 EVA Query Language (EVAQL)

EVA's parser supports a query language tailored for exploratory video analytics, called EVAQL.

Loading Data. EVA allows users to load both unstructured data (*e.g.*, videos and images) and structured data. The following query loads a video into EVA:

```
/* Loading a video into the table */
LOAD VIDEO "videos/*.mp4" INTO VIDEO_DATA;
```

This will automatically create a table named VIDEO_DATA, which includes the following columns: (1) id, (2) data, (3) video_id, (4) video_frame_id, and (5) video_name. These columns represent the frame identifier, the frame's content, and the video to which the frame belongs.

User-Defined Functions. EVAQL is designed to simplify the process of defining user-defined functions (UDFs) that cater to the requirements of different applications in the VDBMS. It supports a wide range of UDFs that take diverse types of inputs (*e.g.*, video meta-data or raw frames *etc.*) and outputs (*e.g.*, labels, bounding boxes, *etc.*).

Users have the option to import their own custom-built UDFs from source, or quickly import an UDF that wraps around a deep learning model from widely-used frameworks (*e.g.*, HuggingFace [18], PyTorch). We next discuss these two options in more detail.

UDF from Source. EVA enables users to define UDFs using Python function *decorators*, allowing them to migrate their deep learning models to EVA with minimal code changes.

```
# Configuring a 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=(1,3,540,540))],
    output_signatures=[PandasDataframe(
                       columns=["label"],
                       column_types=[NdArrayType.STR
                           ])]
    )
    def forward(self): # do inference
```

Using the decorator-based syntax, users define the input and output signatures of their models along with other properties of the UDF. These properties include whether or not EVA should cache the results of the UDF, or whether the UDF supports batch mode execution. The following query registers the UDF in EVA:

```
/* Registering a User-Defined Function */
CREATE UDF ImageClassificationUDF
TYPE  ImageClassification
IMPL '/udfs/image_classification.py'
```

Here, `TYPE` specifies the intended logical type of the UDF (*e.g.*, ImageClassification or ObjectDetection). `IMPL` indicates the path of the UDF implementation.

UDF from HuggingFace. EVA provides out-of-the-box support for HuggingFace tasks and models. Users may quickly define such UDFs using EVAQL, as illustrated in the following query:

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

This command adds a UDF that performs object detection using the `facebook/detr-resnet-50` model.

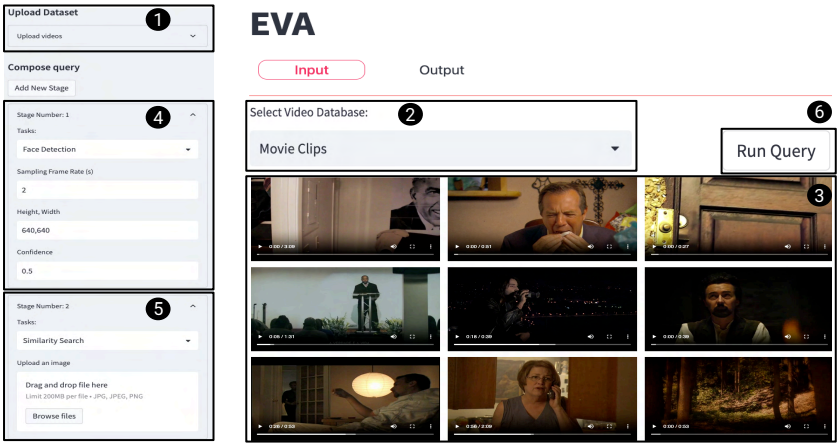

Figure 2: EVA-UI: A visual interface for exploratory video analytics backed by EVA.

2.2 Other Components

Query Optimizer. The EVA's Optimizer is based on the Cascades query optimization framework [9] and applies a series of rules for rewriting the query and performs cost-based optimization to generate a physical query plan. It focuses on minimizing query processing time while meeting the accuracy constraint (which is often not an option in a typical relational DBMS). To guide important optimization decisions, the Optimizer runs vision models on a subset of frames while processing an ad-hoc query [4]. To accelerate exploratory video analytics [19], EVA materializes the results of the expensive UDFs and reuses them while processing subsequent queries.

Execution Engine. The Execution Engine in EVA is responsible for evaluating the query plan generated by the Optimizer, and it leverages heterogeneous computational units such as CPUs and GPUs. The Execution Engine relies on deep learning frameworks, like PyTorch, for model inference. EVA leverages Ray to support distributed query execution. It splits the video data into partitions and uses multiple GPUs for model inference to reduce query processing time. Additionally, EVA supports parallel processing of complex query predicates.

Storage Engine. Lastly, the Storage Engine directly stores videos and images in a compressed format. It manages structured data using Parquet [1] format on disk, and uses the Arrow [16] in-memory columnar format for processing data.

3 DEMONSTRATION

We next describe the three scenarios in the EVA demo. The first scenario presents EVA-UI that offers a user-friendly interface for interactive exploration of video datasets (§ 3.1). In the second scenario, we demonstrate how EVA can be easily used in data science notebooks (§ 3.2). Lastly, in § 3.3, we demonstrate how EVA can serve as a back-end database in other applications, such as Label Studio (§ 3.3). We use the movie dataset [5], the cat and dog dataset [8], and a video feed from a static camera to demonstrate the capabilities of EVA. In each scenario, attendees will be given the opportunity to modify the queries and UDFs.

Figure 3: Integration with Data Science Ecosystem

3.1 Scenario 1: EVA-UI

We utilize Streamlit v1.14.0 and React v15.0 to design the user interface of EVA-UI, which is depicted in Fig. 2. A typical workflow involves six stages: ❶ Uploading a video dataset in the form of an mp4 directory on a local machine or AWS bucket and assigning a name to it. ❷ Selecting the desired dataset from the dropdown menu. ❸ Displaying all videos in the dataset in a panel for easy visualization. The user can view the output by switching to the `output` tab. ❹ Choosing the desired task to perform, such as FaceDetection, with the ability to specify model parameters, including height, width, and confidence. Users can also set a sampling rate (*e.g.*, every 2 secs) to reduce computation. ❺ Creating a multistage query, for example by performing a similarity search on detected faces from stage one. ❻ Executing the query by clicking the `Run` button.

Using this intuitive visual interface, users can execute a complex query, like the query in Listing 1, without learning EVAQL. We note that the user can incrementally add the second stage after completing the first stage. EVA uses UDF caching [19] to automatically utilize the materialized results from the first stage, avoiding the need to rerun the model.

EVA-UI also supports two other workflows. First, users can select a region of interest by drawing one or more bounding boxes over the video frames before executing a task, further reducing the computational cost. Users can also associate specific vision tasks

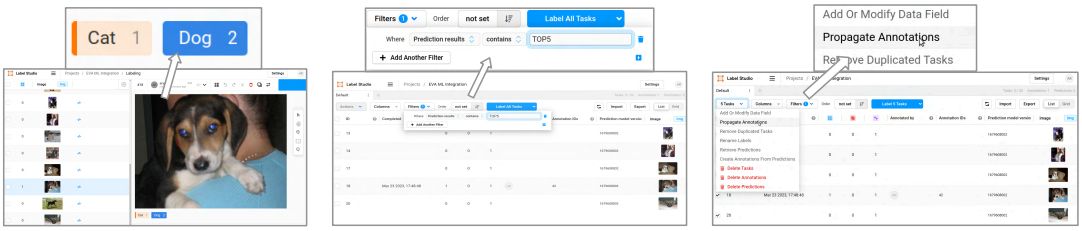

Figure 4: Integration with Label Studio to facilitate bulk labeling

with each region. For example, in a video of a traffic camera, the user can specify one region of interest for a crosswalk and one for the road and detect persons on the crosswalk while a car is also on the road. Second, EVA-UI supports *semantic known entity search*, where users can logically provide the coordinates of different objects in the query. For example, they may specify that they want to find a car in the left half of the frame and a person in the right half. The backend EVA runs object detection and uses spatial predicates to locate the relevant frames in the dataset.

3.2 Scenario 2: Integration with DS Ecosystem

In their daily workflows, data scientists often need to store their data in a data system and process it using custom UDFs. To support this workflow, EVA is available as a Python package in the `pip` package repository. Data scientists can easily import the EVA package in their data science notebooks.

EVA takes care of data storage and query execution, so that the domain scientists may focus more on data analysis. EVA works well with widely-used data science libraries (*e.g.*, Pandas for post-processing, Matplotlib for visualization, and deep learning frameworks like HuggingFace and PyTorch).

To demonstrate the utility of EVA in this scenario, we will showcase the following steps to the audience: ❶ `pip` install EVA and import it into the notebooks, ❷ load unstructured data such as videos or images from various sources (*e.g.*, local filesystem, AWS, or YouTube) into EVA in the notebook, ❸ register a custom UDF written in Python, or import models from popular libraries such as HuggingFace or PyTorch, and ❹ execute a query with multiple UDFs, and obtain the query results as a Pandas dataframe, that is subsequently visualized using Matplotlib.

Fig. 3 illustrates this usage scenario of EVA. Here, EVA is being used to extract the license plate information from a video feed captured by a static camera at an intersection using an OCR model. The user specifies a region of interest and performs fuzzy matching against a target license plate to handle minor model errors.

3.3 Scenario 3: Integration with Label Studio

In the last scenario, we will focus on how EVA complements Label Studio, a popular open-source data labeling framework that enables users to create and manage high-quality labeled datasets for training machine learning models. One of the main challenges with data labeling is the manual effort involved, especially when it comes to labeling large image or video datasets.

In this usage scenario, we illustrate how Label Studio may use EVA as a backend data system for similarity search. This allows EVA to power *bulk label propagation* in Label Studio, so that similar images or objects can be labeled together to reduce human labeling effort. EVA handles the storage and retrieval of labeled images, while also supporting a similarity search UDF for propagating labels to similar images.

Fig. 4 illustrates the bulk labeling workflow enabled in Label Studio, with EVA serving as the backend database system. This example focuses on a labeling task for training an image classification model (cats vs dogs). Images are pre-loaded into Label Studio. Initially, ❶ the user manually assigns a label to an image (*e.g.*, user labels a dog image). Subsequently, ❷ the user utilizes a similarity search UDF based on an off-the-shelf feature extractor (*e.g.*, Resnet) to find the top-K similar images (*e.g.*, K = 10) using EVA. Finally, ❸ the user propagates the label of the initial dog image to all the similar images, using the Label Studio user-interface, thereby saving labeling time.

REFERENCES

[1] 2022-07-01. *Apache Parquet.* https://parquet.apache.org/.
[2] Favyen Bastani and et al. 2020. MIRIS: Fast Object Track Queries in Video. In *SIGMOD*. 1907–1921.
[3] Gary Bradski. 2000. The openCV library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer* 25, 11 (2000), 120–123.
[4] Jiashen Cao et al. 2022. FiGO: Fine-Grained Query Optimization in Video Analytics. In *SIGMOD*. 559–572.
[5] Keith Curtis et al. 2020. HLVU: A New Challenge to Test Deep Understanding of Movies the Way Humans do. In *Proceedings of the 2020 International Conference on Multimedia Retrieval*. 355–361.
[6] Maureen Daum and et al. 2022. VOCAL: Video Organization and Interactive Compositional AnaLytics. In *CIDR*.
[7] Jeff Dean et al. 2018. A new golden age in computer architecture: Empowering the machine-learning revolution. *MICRO* 38, 2 (2018), 21–29. Publisher: IEEE.
[8] Jeremy Elson et al. 2007. Asirra: a CAPTCHA that exploits interest-aligned manual image categorization. *CCS* 7, 366–374.
[9] G. Graefe. 1995. The Cascades Framework for Query Optimization. *IEEE Data Eng. Bull.* 18, 3 (1995), 19–29.
[10] Gaurav Tarlok Kakkar et al. 2023. EVA: An End-to-End Exploratory Video Analytics System. In *Proceedings of the Seventh Workshop on Data Management for End-to-End Machine Learning*. 1–5.
[11] Daniel Kang et al. 2019. BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics. *Proc. VLDB Endow.* 13 (2019), 533–546.
[12] Daniel Kang et al. 2022. VIVA: An End-to-End System for Interactive Video Analytics. In *CIDR*
[13] Marc-André Lemburg. 2001. *Python Database API Specification v2.0.* PEP 249. https://peps.python.org/pep-0249/
[14] Pandas. 2020. *pandas-dev/pandas: Pandas* https://doi.org/10.5281/zenodo.3509134
[15] Adam Paszke and et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*.
[16] Neal Richardson and et al. 2022. *arrow: Integration to Apache Arrow*. https://arrow.apache.org/docs/r/.
[17] Olga Russakovsky and et al. 2015. Imagenet large scale visual recognition challenge. *IJCV* 115, 3 (2015), 211–252. Publisher: Springer.
[18] Thomas Wolf and et al. 2020. Transformers: State-of-the-art natural language processing. In *EMNLP*. 38–45.
[19] Zhuangdi Xu et al. 2022. EVA: A Symbolic Approach to Accelerating Exploratory Video Analytics with Materialized Views. In *SIGMOD*. 602–616.