VIVA: An End-to-End System for Interactive Video Analytics

Daniel Kang*
ddkang@cs.stanford.edu
Stanford University

Francisco Romero*
faromero@stanford.edu
Stanford University

Peter Bailis
pbailis@cs.stanford.edu
Stanford University

Christos Kozyrakis
christos@cs.stanford.edu
Stanford University

Matei Zaharia
matei@cs.stanford.edu
Stanford University and Databricks

ABSTRACT

The growth of video volumes and increased DNN capabilities has led to a growing desire for video analytics. In response, the data analytics community has proposed multiple systems that optimize specific query types (e.g., selection queries) or a particular step in query execution (e.g., video retrieval from storage). However, none of these systems provide *end-to-end, practical* video analytics for users to iteratively and interactively engage with queries, as is the case with analytics systems for structured data.

In response, we are building VIVA: an end-to-end system for interactive video analytics. VIVA contains five novel components. First, VIVA uses *relational hints*, which allow users to express relationships between columns that are difficult to automatically infer (e.g., mentions of a person in a transcript can be used as a proxy for the person appearing in the video). Second, VIVA introduces a *mixed-data query optimizer* that optimizes queries across both structured and unstructured data. Third, VIVA features an *embedding cache* that decides which results/embeddings to store for future queries. Finally, VIVA co-optimizes storage and query execution with its *video file manager* and *accelerator-based execution engine*. The former decides how to pre-fetch/manage video, while the latter selects and manages heterogeneous hardware backends spanning the growing number of DNN accelerators. We describe the challenges and design requirements for VIVA's development and outline ongoing and future work for realizing VIVA.

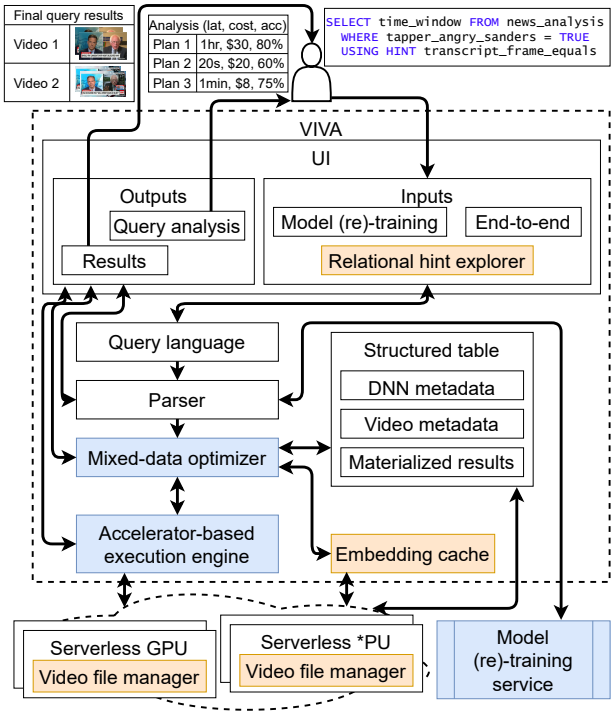


Figure 1: VIVA's architecture diagram. Blue components/interfaces are typically found in analytic systems for structured data that require rethinking. Orange components/interfaces are novel to VIVA.

1 INTRODUCTION

Video volumes are growing tremendously in scale: 500 hours of video are uploaded to YouTube every minute [36]. At the same time, deep neural networks (DNNs) have increased in capabilities, e.g., allowing for detection of objects in videos [17]. These two trends have made automatic and meaningful analyses of video increasingly feasible, allowing users to answer queries such as "how many birds of a particular species visit a feeder per day" or "do any cars that passed an intersection match an AMBER alert."

To date, research systems for DNN-based video analytics focus on optimizing a specific query type (e.g., selecting a frame with a specific criteria) or a single step in query execution. They range from reducing query costs via approximations [21], efficient use of hardware [26], indexing video data [24], to video query programming models [13]. However, users must manually decide how to combine and manage these systems to answer desired queries. Consider an analyst studying political news coverage from the "big three" cable news channels [19]. The analyst may issue several ad-hoc (i.e., exploratory) queries to understand the dataset, such as finding instances of Bernie Sanders (a politician) reacting angrily to Jake Tapper (a TV news host). To answer this query efficiently, the analyst needs to consider several choices, such as: should they train a specialized proxy model for Jake Tapper, Bernie Sanders, or both (e.g., using NoScope [22])? Is it best to filter for Bernie Sanders and Jake Tapper using the video transcripts first before searching frames [13]? Have enough labels (e.g., angry face detections) been materialized to leverage a video event specification system [13]?

Today, end-to-end systems for terabyte-scale structured data analytics (e.g., data warehouses) alleviate the burden on users from having to manually optimize queries over structured data. These systems efficiently automate query planning because structured

*Denotes equal contribution.

 12th Annual Conference on Innovative Data Systems Research (CIDR '22), January 9–12, 2022, Chaminade, USA.

data is in a predictable schema, and the number of query types is limited. We believe similar end-to-end systems will emerge for video analytics. However, designing systems for interactive video analytics (IVA), i.e., ad-hoc queries over large volumes of video data, presents several challenges:

***C1:* Specifying domain knowledge.** To optimize queries, data warehouse systems collect statistics about fields in the tables they store. However, since state-of-the-art DNNs are expensive to execute — as slow as 3 frames per second (fps) — video analytics systems need mechanisms for users to express relationships between unstructured data (i.e., domain knowledge) to optimizers. BlazeIt [21] allows users to query information about videos through virtual relations that trigger the execution of a DNN to materialize the view (e.g., object detection DNN to materialize the label). This allows BlazeIt to lazily optimize the number of DNN invocations. However, since the values of virtual columns are not known until they are materialized, relationships between columns cannot be automatically inferred without guidance from the user.

***C2:* Optimizing across unstructured and structured data.** IVA query costs can be orders of magnitude higher than structured data query costs due to executing expensive DNNs. For example, computing results for a year of video data (60 fps) with a 3fps DNN would cost ~$54K on a 2-core Google Cloud Platform (GCP) instance with NVIDIA T4 [15, 35]. However, IVA systems can reuse query results across users (e.g., faces materialized for one query can be reused for the TV news analysis query). Hence, IVA systems should optimize queries across both structured data (e.g., materialized face labels) and unstructured data (e.g., video frames). Most existing systems optimize either structured [32, 45, 48] or unstructured [6, 21, 33] data, or assume all unstructured data is in embedding vectors (i.e., do not need to pass DNNs over frames in real-time) [46]. IVA systems should also extend video data indexing systems (e.g., TASTI) [24] to decide which results/embeddings to cache or materialize.

***C3:* Efficiently utilizing accelerators for large datasets.** Users are interested in analyzing increasingly large volumes of video. For example, ten years of "big three" US news channels is ~96TB [3]. Thus, assuming all data will be available in a node's local storage is not feasible. Furthermore, efficiently processing these large volumes with DNNs requires accelerators that are expensive to constantly run. Hence, IVA workloads need to run on large-scale, distributed systems with heterogeneous accelerators and data spread through a distributed filesystem. This requires IVA systems to jointly consider storage and compute by (a) strategically laying out video data across distributed storage, and (b) efficiently meeting varying throughput demands across video retrieval, decoding, and DNN execution [47].

To address these challenges, we are building VIVA (Figure 1), an *end-to-end* IVA system. In this work we highlight the key ideas of our design. In particular, VIVA contains several novel components:

***S1:* Describing domain-specific relations.** VIVA allows users to express domain knowledge through *relational hints*. Relational hints generalize the notion of a proxy model. They are a declaration of how two virtual columns (i.e., columns materialized by a DNN's output) are related to each other. For example, a user can describe that all Bernie Sanders faces are a superset of his angry faces. Furthermore, as users explore query strategies based on different relational hints, VIVA provides interactive feedback — latency, accuracy, cost analyses, and samples of resulting frames and video clips — so users can visually explore their query results and tune its quality through relational hints.

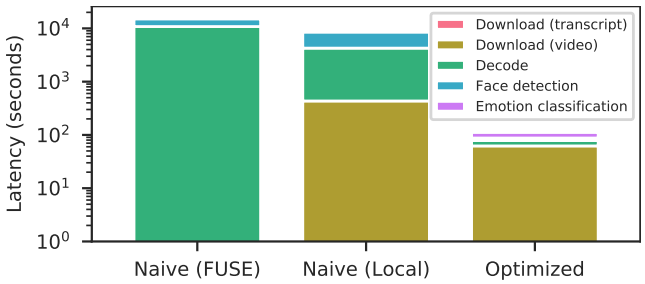


Figure 2: Latency breakdown of three query plans for angry Bernie Sanders interviews. Y-axis is log-scaled. `Naive (FUSE)` and `Naive (Local)` execute face recognition and emotion classification over all frames. `Optimized` filters using the transcript, then executes the face recognition and emotion classification on filtered frames with pipelining. The resource-dominant step varies by query plan.

***S2:* Blending structured and unstructured data.** VIVA's *mixed-data query optimizer* spans both structured records and unstructured records. The optimizer queries structured tables and an *embedding cache* to determine operation ordering, what pre-computed results or embeddings are available, and to provide the user with exploratory cost estimates for query plans.

***S3:* Co-optimizing storage and compute.** VIVA's *video file manager* pre-fetches, caches, and manages video data (e.g., selecting resolutions or tile layouts [12]). VIVA's *accelerator-based execution engine* selects from and manages heterogeneous hardware backends that span the growing number of DNN accelerators, especially as they become available on serverless offerings.

We explore how the highlighted challenges have guided VIVA's design, show preliminary results supporting our design choices, and outline ongoing and future work for realizing VIVA.

2 FEASIBILITY OF IVA

To understand the feasibility and unresolved challenges for IVA, consider three query plans for the TV news analysis query: (a) `Naive (FUSE)`: mounting remote object storage with FUSE [14], directly decoding frames from the mounted drive, and executing face recognition followed by emotion classification, (b) `Naive (Local)`: same as (a), but first downloading the videos from remote storage before decoding the local videos, and (c) `Optimized`: filtering frames via the transcript, then executing the remainder of (b) on the filtered frames and pipelining results across steps.

To compare these three plans, we manually combined existing techniques as users would do today. Each was run on 100 GCP `n1-standard-16` with one NVIDIA T4 GPU over one year of CNN videos with results in Figure 2. As shown, `Naive (FUSE)` would take ~4.2 hours and ~$325, `Naive (Local)` would take ~2.5 hours and ~$200, and `Optimized` would take ~3 minutes and ~$4. Thus, IVA systems have the potential to deliver fast and cheap ad-hoc queries over large collections of videos.

There are several challenges towards achieving or improving on the performance and cost of `Optimized`: exploiting domain knowledge (e.g., using transcripts for filtering), automatically selecting the right optimizations across both structured data and unstructured data, and managing video data processing across hundreds to thousands of workers with heterogeneous accelerators (including optimizing video retrieval). Overcoming these challenges would benefit several domains including performing large-scale ecological science, enabling warehouse-scale robotics deployment and analysis, and curating data for autonomous vehicles [9, 28].

3 VIVA

To address the challenges in Section 2, we are building VIVA: an end-to-end IVA system. We first describe the query types supported by VIVA (Section 3.1). Next, we give a detailed workflow overview of how an analyst would interact with VIVA (Section 3.2). Finally, we describe the novel components in VIVA's architecture that build on and extend prior work in video analytics (Section 3.3).

3.1 Query Types Supported by VIVA

VIVA supports five common classes of data analytics queries: selection, aggregation, limit, similarity, and join. A significant difference in processing these classes of queries for IVA workloads is the cost of materialization due to executing expensive DNNs. This cost can be orders of magnitude more expensive than materializing views for a standard database [21]. We describe each query type from a video analytics perspective and discuss its implications for VIVA.

Selection queries. Users are interested in selecting particular objects or events of interest. Analysts may search for instances of Bernie Sanders reacting angrily to Jake Tapper or diving header goals in soccer games. These are often used for downstream, manual analysis. As such, they are relatively rare (<5% prevalence) [23].

Aggregation queries. Users are interested in computing some statistic over the video frames. A city planner may compute the average number of cars per frame or count the number of pedestrians that cross in front of a doorbell camera. Since many applications tolerate approximations, VIVA can apply sampling techniques.

Limit queries. Users are interested in finding a cardinality-limited number of events for manual inspection or for correctness tests. A city planner may search for 10 instances of buses at stop signs or an analyst may search for two instances of diving header goals.

Similarity queries. Users are interested in searching for portions of the video similar to an input frame or video clip. For example, a football analyst may input a frame or video clip of an offensive formation scheme they wish to search for. Such queries often involve iterative, ad-hoc analysis to arrive at the final query.

Join queries. Users are interested in performing a join and subsequently performing a selection, aggregation, or limit query. An AMBER Alert application may join extracted license plates with an external data source. Often, extracting the join column is expensive. While VIVA can naively answer such queries, we defer optimizations to future work.

3.2 Workflow Example

In this section, we describe how an analyst would use VIVA for a selection query (Section 3.1) to search for interview scenes with Jake Tapper and angry Bernie Sanders. Table 1 shows an abbreviated schema for a TV news analysis table, `news_analysis`.

With VIVA, the analyst can begin by defining a column, `face_label`, produced by calling a user-defined function (UDF) that passes a face detection DNN over a frame:

```
ALTER TABLE news_analysis ADD face_label
  AS face_detection(frame)
```

Similarly, the analyst can define a column, `sentence_segment`, produced by a UDF that gets to get sentence segments corresponding to a frame:

```
ALTER TABLE news_analysis ADD sentence_segment
  AS get_sentence_segment(frame)
```

Next, the analyst can define columns specific to their query. To produce the column `sanders_frame`, the analyst would input:

```
ALTER TABLE news_analysis ADD sanders_frame
  AS face_label = 'Sanders'
```

A similar expression would produce `tapper_frame`. To capture emotions, the analyst defines a column, `emotion_label`, produced by a UDF that passes an emotion detection DNN over a frame:

```
ALTER TABLE news_analysis ADD emotion_label
  AS emotion_detection(frame)
```

Similar to `sanders_frame`, the analyst can define a column, `sanders_angry`:

```
ALTER TABLE news_analysis ADD sanders_angry
  AS sanders_frame = TRUE AND emotion_label = 'angry'
```

Finally, the analyst can define a column, `tapper_angry_sanders`, corresponding to their end query for finding interview scenes with Jake Tapper and angry Bernie Sanders:

```
ALTER TABLE news_analysis ADD tapper_angry_sanders
 AS sanders_frame = TRUE
   AND tapper_frame = TRUE
   AND sanders_angry = TRUE
```

The analyst would then specify the end query as:

```
SELECT time_window FROM news_analysis
  WHERE tapper_angry_sanders = TRUE
```

They can then optionally receive query cost insights by augmenting their query with the EXPLAIN statement.

The analyst will notice it is slow and cost-inefficient to pass DNNs over all frames to detect Bernie Sanders. Thus, they can explore an alternative way of detecting Bernie Sanders by using transcripts. To create the column `sanders_transcript`, the analyst would input:

```
ALTER TABLE news_analysis ADD sanders_transcript
  AS sentence_segment LIKE '%Sanders%'
```

Next, the analyst would explore the results of searching for "Sanders" in transcripts (`sanders_transcript`) by inputting:

```
SELECT time_window FROM news_analysis
  WHERE sanders_transcript = TRUE
```

Field	Type	How field is generated
channel	string	Video metadata
time_window	(float,float)	Video metadata, DNN
face_label	string	Object detection DNN
emotion_label	string	Emotion detection DNN
sentence_segment	string	Sentence from transcript
sanders_frame	bool	Computed column for Sanders in frame
tapper_frame	bool	Computed column for Tapper in frame
sanders_transcript	bool	Computed column for Sanders in transcript
sanders_angry	bool	Computed column for Sanders being angry
tapper_angry_sanders	bool	Computed column for interview scenes with Tapper and angry Sanders

Table 1: Table schema for TV news analysis table, `news_analysis`. A query for Bernie Sanders scenes would use `sanders_frame`, while Section 2's TV news analysis would use `tapper_angry_sanders`.

From the results, the analyst will notice detecting Bernie Sanders through transcripts is orders of magnitude faster than detecting him in frames. However, it is also less accurate. Using this knowledge, they can express a relational hint to VIVA that transcript segments in which Bernie Sanders speaks are a proxy for frames he appears in (i.e., a general hint, see Section 4.2):

```
CREATE GENERAL HINT transcript_frame_proxy
  FOR news_analysis
  AS sanders_transcript PROXY sanders_frame
```

After inspecting the results, the analyst will further note that for interview queries, transcript segments in which Bernie Sanders speaks are equivalent to frames he appears in. Thus, the analyst can specify a relational hint that only holds for their query (i.e., an explicit hint, see Section 4.2) as follows:

```
CREATE EXPLICIT HINT transcript_frame_equals
  FOR news_analysis
  AS sanders_transcript EQUALS sanders_frame
```

Once the hints are defined, the analyst can specify the end query to the end-to-end interface with the relational hint specific to this query (`transcript_frame_equals`). VIVA will efficiently execute the query over the entirety of the video corpus, and will automatically leverage the general relational hint, `transcript_frame_proxy`:

```
SELECT time_window FROM news_analysis
  WHERE tapper_angry_sanders = TRUE
  USING HINT transcript_frame_equals
```

During the exploration VIVA materializes and caches results (e.g., Bernie Sander's faces) to speed up subsequent queries (Section 5.4). Furthermore, since query cost can be data-dependent (e.g., the cost can depend on how many faces are present in a frame), VIVA collects statistics (e.g., average number of objects per frame) as queries execute. VIVA incorporates these statistics into its query planning and cost estimation similar to existing systems that sample based on a user-inputted accuracy target, e.g., systems that use statistics to decide how to allocate samples for stratified sampling [25].

3.3 Key VIVA Components

We now discuss the key components of VIVA's architecture — shown in Figure 1 — that enables it to address the challenges described in Section 2. Orange components are novel, while blue components are typically found in analytics systems (e.g., RDBMSes) that require redesigning.

Describing domain-specific relations. VIVA's *relational hints explorer* allows users to explore their video corpus and express domain-specific relationships through relational hints. To aid users in their exploration, VIVA provides interactive feedback (similar to SQL's EXPLAIN) — expected cost, latency, and accuracy of a query, or a sample of the resulting frames and video clips for the user to visually assess (e.g., over a subset of the dataset). By providing this interactive feedback, users can explore variations of queries that achieve the same end goal. For example, a user might explore the results of finding Bernie Sanders in different ways: (a) using transcripts, (b) using a general face recognition DNN, or (c) using a specialized model trained to detect Bernie Sanders.

Blending structured and unstructured data. The *mixed-data query optimizer* (Sections 4.1 and 5.2) optimizes query plans by considering both unstructured records (e.g., video frames) and structured records (e.g., structured table in Figure 1). For unstructured records, VIVA extends and builds upon existing work on reducing query cost (e.g., approximations and object tracking). VIVA also features an *embedding cache* that extends existing work on video indexing (e.g., TASTI [24]) to decide which results/embeddings to store for future queries (Section 5.4).

Co-optimizing storage and compute. Each compute instance has a *video file manager* that fetches video files from remote storage (e.g., low-cost systems like Google Cloud Storage) or distributed nodes. VIVA's video file manager is motivated by work like TASM and VStore, which focus on optimizing how locally-available video data is formatted (e.g., resolution) or tiled. However, the video file manager also decides whether frames or chunks should be cached or pre-fetched. VIVA's *accelerator-based execution engine* manages the heterogeneous compute instances for executing queries. Based on the DNN and underlying accelerator selected by the mixed-data query optimizer, the accelerator-based execution engine makes scheduling and resource allocation decisions for decoding and compute. We describe both components in Section 5.3.

4 SPECIFYING RELATIONAL HINTS

To optimize queries, systems must understand the relationships between columns. Some relationships can be inferred using existing functional dependency techniques [20, 29]. Unfortunately, many relationships cannot automatically be inferred, e.g., it is difficult to automatically infer that searching for Bernie Sanders interviews with Jake Tapper should not involve cooking shows. Existing work on functional dependencies is limited to structured data, while IVA workloads span both structured data and unstructured data.

Similar to how relational databases support foreign keys to link structured data between two tables, we propose relational hints for virtual columns. Relational hints generalize the notion of a proxy model [21, 22]: they are a declaration of how two virtual columns are related to each other. They allow users to specify domain knowledge to VIVA. Implicit in all of the hints is a time window for which the hints apply over (omitted for brevity in the description of the relational hints).

4.1 Relational Hints Operators

VIVA currently supports four relational hints operators that cover queries across a wide range of use cases (see Section 2); other hints can be added for future use cases: SUPERSET, EQUALS, EXCLUDES, and PROXY. Throughout, *A* and *B* refer to columns, some of which may be virtual and some of which may be materialized. A column can either correspond to a single frame, or a group of frames (e.g., a time window over which a label is true).

SUPERSET. *A SUPERSET B* denotes that *A* is a superset of *B*. For example, Bernie Sanders present in a frame (`sanders_frame`) is a superset of Jake Tapper interviewing an angry Bernie Sanders (`tapper_angry_sanders`).

Given *A SUPERSET B* and a query for *B*, VIVA will produce a query plan where *B* only processes data that correspond to columns predicated on *A*. If *A* is not materialized, VIVA will produce a query plan to first process *A*. Subsequent cost modeling will determine which plan to execute.

EQUALS. *A EQUALS B* denotes that *A* and *B* are equivalent. For example, an analyst may express that Bernie Sanders in the transcript (`sanders_transcript`) is equivalent to Bernie Sanders present in a frame for a query searching for Bernie Sanders being interviewed.

Given *A EQUALS B* and a query for *A* or *B*, VIVA will produce two query plans using the other column. Similar to SUPERSET, VIVA will produce a query plan materializing *A* if it is not already.

EXCLUDES. *A EXCLUDES B* denotes that if *A* is true, *B* is false. For example, Jake Tapper interviewing Bernie Sanders will not happen on a cooking show.

Given *A EXCLUDES B* and a query for *B*, VIVA will produce a query plan that does not process any data for which *A* is true. Similar to EQUALS and SUPERSET, if *A* is not materialized, VIVA will produce a query plan to first process *A*.

PROXY. *A PROXY B* denotes that *A* can be used to approximate predicates for *B*. For example, Bernie Sanders in the transcript is a proxy for Bernie Sanders present in a frame.

Given *A PROXY B*, and a user query for *B* with an accuracy target, VIVA will apply techniques for approximation (e.g., BlazeIt). Since proxies are inherently noisy, they are used in approximate queries. VIVA currently supports existing techniques such as approximate selection, aggregation, and track processing [6, 21–23, 25].

4.2 Relational Hints Scope

As shown in Section 3.2, an analyst may find that (a) relationships between columns hold for any query over a schema, or (b) relationships are only valid for certain queries. To allow the expression of both types of relationships in queries, VIVA supports two types of hints: general and explicit hints.

General hints. These hints can be reused across queries and hold for any query over the schema. For example, the relational hint specifying Bernie Sanders in the transcript is a PROXY for Bernie Sanders present in a frame in Section 3.2 is a general hint. General hints can be registered by any user and can be reused across queries. VIVA automatically uses registered relational hints when applicable. We envision general relational hints being useful in a collaborative setting, where users with domain knowledge can specify relational hints that teammates can leverage without having to redefine them.

Technique	Result
Decode	Read video into uncompressed raw format [40]
Encode	Compress raw video into a target format [40]
Frame pre-processing	Apply frame manipulations (e.g., cropping) [26]
Face detector	Label frames with faces [1]
People detector	Label frames with people [1]
Vehicle detector	Label frames with vehicles [1]
OCR	Identify text in frame [39]
Labeling model	Use high accuracy general model to label and train a domain specific model [21]
ASR	Transcribe video audio to text [39]

Table 2: Examples of techniques VIVA applies at video ingestion.

Explicit hints. Explicit hints are not applied unless requested. For example, the hint specifying Bernie Sanders in the transcript EQUALS Bernie Sanders being present in a frame in Section 3.2 is an explicit hint. While true for interviews, it does not generally hold. Thus, explicit hints are not automatically reused across queries.

5 END-TO-END IVA QUERY EXECUTION

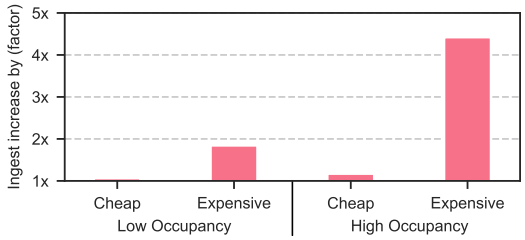
During query execution, VIVA not only optimizes over structured and unstructured data, but also decides when to materialize results. In this section, we explain how VIVA processes video data by first describing the data VIVA materializes at ingest to enable interactive video queries (Section 5.1). At query time, VIVA's mixed-data query optimizer considers both structured and unstructured data and incorporates relational hints (Section 5.2). During execution, VIVA co-optimizes storage and compute across heterogeneous compute backends (Section 5.3). Once the query completes, VIVA decides what results/embeddings to cache (Section 5.4).

5.1 Materializing Results at Video Ingest

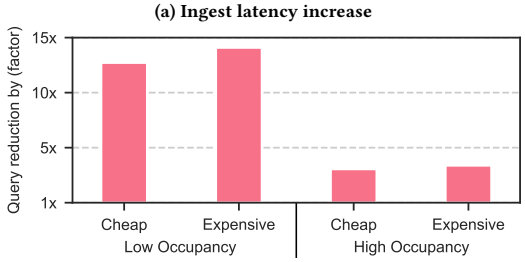
During ingestion, videos are transcoded and stored for later viewing across various types of devices [40]. Since transcoding requires touching each frame of a video [30], VIVA can materialize commonly-used views at ingestion time. In particular, the cost of executing certain DNNs combined with sampling, e.g., detecting a sample of faces across ten years of "big three" news coverage, is marginally low compared to the cost of transcoding.

To reduce the number of frames processed at query time, VIVA leverages ingest techniques developed in prior work and deployed in various academic and industry systems (examples shown in Table 2). These state-of-the-art techniques trade-off performance and accuracy (e.g., Smol [26]). VIVA uses occupancy (prevalence of a given object class in a video [21]) to determine which frames to process at ingest. While materializing views at ingest generally benefits all query types (see Section 3.1), there may be additional considerations in some cases. For example, similarity queries may require more temporal information (i.e., frames to process) to establish a relationship (see Section 7).

We consider a selection query to find all occurrences of a given object in a video under two scenarios: `Cheap` ingest, which runs all techniques from Table 2 except ASR and OCR, and `Expensive` ingest, which runs all the techniques. Running ASR and OCR extracts



(a) Ingest latency increase



(b) Query latency reduction

Figure 3: Normalized ingest and query latencies of VIVA. We consider two scenarios. `Cheap` ingestion: run all techniques from Table 2 except ASR and OCR. `Expensive` ingestion: run all techniques from Table 2. VIVA trades off the time and cost spent extracting results at video ingest to make interactive video queries feasible.

more information at ingest and can further reduce query latency while incurring higher ingest latency. We consider two types of video settings: low occupancy where object occurrence in frames is low and high occupancy where object occurrence is high (e.g., traffic cams at rural and urban intersections, respectively).

Figure 3 shows the results of this evaluation. Ingest and query latencies are normalized to a baseline of no materialized results. In the low occupancy scenario, we can reduce the query latency by 12.7× (Figure 3b) for a small ingest increase of 6% (Figure 3a). This is expected because the fast people, vehicle, and face detectors filter a large amount of frames at ingest (over a 5× reduction in frames to process over the baseline). The `Expensive` ingest further reduces query latency, but the ingest cost is noticeably higher compared to `Cheap`. In the high occupancy scenario, the savings are not as significant. Interestingly, the ingest latency for the `Expensive` ingest is significantly higher than the low occupancy scenario. In this case, the more expensive models (ASR and OCR) need to run over a larger number of frames at ingest, but the query latency reduction over `Cheap` is minimal because the expected savings from these models is relatively small. Thus, materializing results at ingest can reduce query latency.

5.2 Blending Structured and Unstructured Data

Given a user query and its relational hints, VIVA produces one or more candidate query plans. VIVA's mixed-data optimizer estimates the cost of these candidates, and selects the most cost-effective one (e.g., best performance/$). The mixed-data optimizer considers execution cost over both structured records and unstructured records (e.g., executing a DNN over a frame). For structured records, the mixed-data optimizer considers execution cost similar to existing database optimizers. For unstructured records, the mixed-data optimizer considers the cost of executing DNNs.

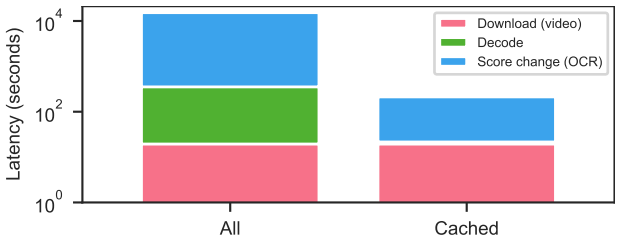


Figure 4: Latency breakdown of two query plans for field goals in NFL games. Y-axis is log-scaled. `All` uses OCR to detect score changes on all frames, then filters by changes of three points. `Cached` uses cached score changes, then uses OCR to detect score changes of three points. VIVA improves performance by 72×.

Similar to existing heterogeneous-aware DNN scheduling systems (e.g., Llama [42], INFaaS [41], Gavel [34]), the mixed-data optimizer uses the most up-to-date backend resource availability to select between candidate query plans. For example, for *A SUPERSET B*, if neither *A* nor *B* is materialized VIVA will use backend resource availability to decide which is cheaper to execute. When multiple DNN candidates are available to run for a given UDF (e.g., multiple face recognition models), the mixed-data optimizer analyzes cost, latency, and accuracy given the available hardware platforms.

5.3 Co-optimizing Execution & Video Fetch

Since IVA workloads are ad-hoc and require variable amounts of compute and memory resources, the serverless computing model (i.e., fine-grained billing for the duration of a query plan's execution) is a good fit [37]. However, IVA workloads require heterogeneous accelerators to efficiently run DNNs. At the time of writing, commercial serverless offerings (e.g., AWS Lambda and Azure Functions) do not support accelerators. Thus, VIVA manages its own serverless workers across heterogeneous accelerators, and leverages optimizations for jointly optimizing pre-processing and inference like Smol [26].

The time to fetch video chunks can be a limit to achieving interactive performance (Figure 2): to complete the angry Sanders query in under ten seconds, we estimate the sustained bandwidth between *each* compute node (`n1-standard-16` with one NVIDIA T4 GPU) and remote storage would need to be ~2.5 GB/s (20 Gbps). This is the peak bandwidth of an SSD or about 4 hard disks [5]. In addition, depending on their storage format, video decoding and transcoding can also be performance limiters. Thus, intelligently deciding what videos to pre-fetch and cache on a compute node is important for both performance and resource efficiency. To do so, each workers' video file manager manages video file accesses and storage formatting. For example, based on exploratory TV news analysis queries, VIVA pre-fetches video data from CNN, decodes the videos, and even tiles frames if labels are available. When the analyst submits the end query over a year of CNN videos, VIVA will have already optimized video downloading and decoding, and materialized some or all of the columns needed (possibly during video ingestion, see Section 5.1).

5.4 Caching and Materializing Results

After a query executes, VIVA caches the outputs and adds any materialized results to the structured table. The mixed-data optimizer uses cached results to accelerate future queries. Storing materialized views is cheap compared to the cost of storing video: face labels for 10 years of TV news is ~5GB, compared to 32TB of storage for the videos (<0.01% overhead). Thus, VIVA exhaustively caches results.

Caching materialized results can result in large query speedups. We use a prototype of VIVA with the embedding cache to query for all made field goals (three-point kicks) over an NFL season. Similar to the example from Section 2, we use 100 GCP `n1-standard-16` each with one NVIDIA T4 GPU. Figure 4 shows the latency of executing this query over video data of one NFL season in two different ways: (a) `All`: uses optical character recognition (OCR) over all frames to detect score changes, then filter by changes of three points, and (b) `Cached`: starting with cached results of when score changes occur, uses OCR to detect the amount by which score changed, then filter by changes of three points. By caching results, VIVA improves query performance by 72×.

TASTI [24] proposes methods for training video indexes that can be cached for reuse across queries. VIVA extends this work by managing and tracking the DNN version (and its associated accuracy) that produce the results. This allows VIVA to leverage the cached results even if the DNN is updated or changed. For example, together with the stored DNN metadata, VIVA can use previously-cached results as a proxy for the updated DNN.

6 RELATED WORK

While recent work has looked at specific aspects to improve cost for IVA queries, VIVA is the first to combine them end-to-end.

Proxies. Recent work uses cheap approximations to accelerate specific classes of queries, ranging from selection [22, 23], aggregation [21], and aggregation with predicates [25]. While these systems can accelerate individual queries, they are focused on the batch setting and only accelerate individual queries. We leverage this line of work for accelerating certain classes of queries in VIVA.

Execution engines. Other work optimizes execution of DNNs [38, 41, 49]. These systems aim to efficiently use hardware resources for already-specified execution plans. However, they do not optimize queries end-to-end and are not typically built for interactive video analytics. Several ideas from these systems can be used in VIVA's accelerator-based execution engine, as described in Section 3.3.

Storage and decoding. Classical (e.g., multimedia databases [2]) and recent work [12, 16, 26, 47] have focused on the storage and decoding of video data, which is increasingly becoming the bottleneck for certain queries via optimized analytics systems. This work aims to optimize storage costs and DNN execution, including preprocessing video data. We leverage ideas from these systems with VIVA's video file manager.

Specifying video queries. Recent work focuses on specifying video queries, either with fixed schemas [21] or for ad-hoc queries [13]. Much of this work focuses on specifying query languages for fixed schemas, such as finding/counting specific object types [21] or tracks [6]. Other work allows users to specify ad-hoc queries [13], but assumes the expensive DNNs have been exhaustively executed over the data, which is infeasible for many organizations.

Interactive analytics. Classical and recent work aims to enable interactive analytics (e.g., online aggregation gives increasingly accurate answers to aggregation queries as queries execute [18]). More recent work focuses on interactive "web-scale" tabular data [32], tabular ML pipelines [11], and live queries [7]. While these techniques do not directly apply to unstructured data, VIVA uses scale-out, data layout, and other structured data optimizations.

Functional dependencies. Existing work on functional dependencies [20, 29] helps database designers automatically determine the relation of one attribute to another. However, existing work is limited to structured data; IVA workloads need to interact with both structured records and unstructured records. VIVA can leverage this work to automatically infer hints for structured data.

7 DISCUSSION AND RESEARCH DIRECTIONS

While our preliminary prototype and results with VIVA have been promising, we have come across the following open questions for the design of end-to-end IVA systems.

Inferring and proposing relational hints. Automatically inferring relational hints (Section 4) is not always possible, especially if they are explicit hints. Furthermore, since virtual columns are not always materialized, standard techniques for detecting functional dependencies cannot directly be applied.

If relational hints are not specified, VIVA warns the user if a query is estimated to be expensive based on the amount of unstructured data that needs to be processed and the required DNNs. We are investigating how query- or domain-specific knowledge can be used to suggest or distill hints.

Supporting cost and latency targets. VIVA allows users to explore their queries and receive insights into cost, latency, and accuracy. However, users may wish to limit a query's execution time or cost. Llama [42] give users the ability to input a cost or latency target for their statically-declared pipeline. However, Llama's optimizations to meet a user's target only span unstructured data and rely on the user to specify the end-to-end pipeline prior to execution. We are investigating how to extend Llama to include structured data, which can bound how many frames to process.

Model (re)-training. Prior work [21, 22] has shown that proxy models can be trained and used to accelerate certain types of queries (e.g., aggregate and limit). However, it is currently left to users to (a) decide when these models should be trained (or re-trained), and (b) select training data to use.

Ideally, VIVA will be able to propose when a DNN should be (re)-trained based on the input query's requirements and the available labels. In particular, VIVA should be able to detect when new data may have changed (e.g., using model assertions [27] or ensemble learning [10]). However, giving a model a "new capability" requires further investigation. First, how can domain knowledge be communicated to the model re-training system? Second, how can the system collect contrastive positive and negative examples from a dataset, especially in the cases that events are rarely occurring? Adding interactivity to model training [8] and label selection may

help to create new detectors for challenging queries (e.g., searching for autonomous vehicle interactions at stop signs [31]). If not done efficiently, however, the human can become the bottleneck.

Understanding the cost of materializing at video ingest. Section 5.1 showed there are tradeoffs in the techniques run at ingest, with higher ingest costs resulting in lower query times. Extracting *potentially* meaningful information many require several models to be run, which can result in a high ingest cost. Naively running detectors at ingest may result in materialized results that do not contribute to interactive latencies. Thus, choosing the right ingest techniques and DNNs to minimize query latency and cost for providers and users is an open research question.

Infrastructure for high throughput demands. VIVA has varying and occasional high demands for throughput across storage, decoding, and compute (Figure 4). This makes building scalable and balanced datacenter infrastructure to support IVA workloads difficult. Thus, researchers should explore options to support this high demand, including video transcoding accelerators [40, 44], accelerators close to storage [4], and hardware disaggregation [43].

Supporting other unstructured data types. For unstructured data, VIVA currently supports video-related records such as frames, transcripts, and audio. However, there are other forms of telemetry data (e.g., LIDAR sensors on autonomous vehicles) that can aid in more efficiently processing video analytics queries. For example, understanding the depth of objects may be expensive or infeasible to determine using only video data. Given the growing pervasiveness of these sources of telemetry data, it makes sense to explore how VIVA can incorporate them into its workflow.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their helpful feedback. We also thank Johann Hauswald for his contributions to VIVA's work on materializing results at ingest time, and Kostis Kaffes, Pratiksha Thaker, and Deepak Narayanan for their insightful discussions to improve this work. This work was supported by the Stanford Platform Lab and its industrial affiliates (Cisco, Facebook, Google, Nasdaq, NEC, VMware, and Wells Fargo). It was also supported in part by affiliate members and other supporters of the Stanford DAWN project — Ant Financial, Facebook, Google, and VMware — as well as Cisco, SAP, and the NSF under CAREER grant CNS-1651570. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Francisco Romero is supported by a Stanford DARE Fellowship, and Daniel Kang is supported by a Google PhD Fellowship.

REFERENCES

[1] 2021. NVIDIA DeepStream SDK. https://developer.nvidia.com/deepstream-sdk
[2] Donald A. Adjeroh and Kingsley C. Nwosu. 1997. Multimedia Database Management —Requirements and Issues. *IEEE MultiMedia* (1997).
[3] Internet Archive. 2021. TV News Archive. https://archive.org/details/tv.
[4] AWS. 2021. AQUA (Advanced Query Accelerator) for Amazon Redshift. https://aws.amazon.com/redshift/features/aqua/.
[5] Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan. 2018. *The Datacenter as a Computer: Designing Warehouse-Scale Machines, Third Edition.*
[6] 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.
[7] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R. Madden, Fred Reiss, and Mehul A. Shah. 2003. TelegraphCQ: Continuous Dataflow Processing. In *SIGMOD*.
[8] Mayee F. Chen, Daniel Y. Fu, Frederic Sala, Sen Wu, Ravi Teja Mullapudi, Fait Poms, Kayvon Fatahalian, and Christopher Ré. 2020. Train and You'll Miss It: Interactive Model Iteration with Weak Supervision and Pre-Trained Embeddings. arXiv:2006.15168 [stat.ML]
[9] Sandeep P. Chinchali, Evgenya Pergament, Manabu Nakanoya, Eyal Cidon, Edward Zhang, Dinesh Bharadia, M. Pavone, and S. Katti. 2020. HarvestNet: Mining Valuable Training Data from High-Volume Robot Sensory Streams. In *ICRA*.
[10] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *NSDI*.
[11] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2015. Vizdom: Interactive Analytics through Pen and Touch. *PVLDB* (2015).
[12] Maureen Daum, Haynes Brandon, Dong He, Amrita Mazumdar, and Magdalena Balazinska. 2021. TASM: A Tile-Based Storage Manager for Video Analytics. In *ICDE*.
[13] Daniel Y Fu, Will Crichton, James Hong, Xinwei Yao, Haotian Zhang, Anh Truong, Avanika Narayan, Maneesh Agrawala, Christopher Ré, and Kayvon Fatahalian. 2019. Rekall: Specifying video events using compositions of spatiotemporal labels. *arXiv preprint arXiv:1910.02993* (2019).
[14] Google. 2021. gcsfuse: A user-space file system for interacting with Google Cloud Storage. https://github.com/GoogleCloudPlatform/gcsfuse.
[15] Google. 2021. Google Compute Engine: GPUs Pricing. https://cloud.google.com/compute/gpus-pricing.
[16] 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/PODS*.
[17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *ICCV*. IEEE, 2980–2988.
[18] Joseph M Hellerstein, Peter J Haas, and Helen J Wang. 1997. Online aggregation. In *Acm Sigmod Record*, Vol. 26. ACM, 171–182.
[19] James Hong, Will Crichton, Haotian Zhang, Daniel Y Fu, Jacob Ritchie, Jeremy Barenholtz, Ben Hannel, Xinwei Yao, Michaela Murray, Geraldine Moriba, et al. 2020. Analyzing Who and What Appears in a Decade of US Cable TV News. *arXiv preprint arXiv:2008.06007* (2020).
[20] Ihab F. Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. 2004. CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies. In *SIGMOD*.
[21] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics. *PVLDB* (2019).
[22] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: optimizing neural network queries over video at scale. *PVLDB* (2017).
[23] Daniel Kang, Edward Gan, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2020. Approximate Selection with Guarantees using Proxies. *PVLDB* (2020).
[24] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2020. Task-agnostic Indexes for Deep Learning-based Queries over Unstructured Data. *arXiv preprint arXiv:2009.04540* (2020).
[25] Daniel Kang, John Guibas, Peter Bailis, Yi Sun, Tatsunori Hashimoto, and Matei Zaharia. 2021. Accelerating Approximate Aggregation Queries with Expensive Predicates. *PVLDB* (2021).
[26] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia. 2021. Jointly optimizing preprocessing and inference for DNN-based visual analytics. *PVLDB* (2021).
[27] Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. 2020. Model Assertions for Monitoring and Improving ML Models. 2 (2020).
[28] Fiodar Kazhamiaka, Matei Zaharia, and Peter Bailis. 2021. Challenges and Opportunities for Autonomous Vehicle Query Systems. In *CIDR*.
[29] Sebastian Kruse and Felix Naumann. 2018. Efficient Discovery of Approximate Dependencies. *Proc. VLDB Endow.* (2018).
[30] Andrea Lottarini, Alex Ramirez, Joel Coburn, Martha A. Kim, Parthasarathy Ranganathan, Daniel Stodolsky, and Mark Wachsler. 2018. Vbench: Benchmarking Video Transcoding in the Cloud. In *ASPLOS*.
[31] Matroid. 2020. AI for Full-Self Driving at Tesla. https://www.youtube.com/watch?t=513&v=hx7BXih7zx8&feature=youtu.be.
[32] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. 2010. Dremel: Interactive Analysis of Web-Scale Datasets. *Proc. VLDB Endow.* (2010).
[33] Oscar R. Moll, Favyen Bastani, Sam Madden, Mike Stonebraker, Vijay Gadepally, and Tim Kraska. 2020. ExSample: Efficient Searches on Video Repositories through Adaptive Sampling. *CoRR* abs/2005.09141 (2020).
[34] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. 2020. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. In *OSDI*.

[35] NVIDIA. 2021. NVIDIA T4. https://www.nvidia.com/en-us/data-center/tesla-t4/.
[36] Oberlo. 2021. Youtube Statistics. https://www.oberlo.com/blog/youtube-statistics.
[37] Matthew Perron, Raul Castro Fernandez, David DeWitt, and Samuel Madden. 2020. Starling: A Scalable Query Engine on Cloud Functions. In *SIGMOD*.
[38] Alex Poms, William Crichton, Pat Hanrahan, and Kayvon Fatahalian. 2018. Scanner: Efficient Video Analysis at Scale (To Appear). (2018).
[39] An Qin, Mengbai Xiao, Yongwei Wu, Xinjie Huang, and Xiaodong Zhang. 2021. Mixer: efficiently understanding and retrieving visual content at web-scale. *Proc. VLDB Endow.* (2021).
[40] Parthasarathy Ranganathan et al. 2021. Warehouse-Scale Video Acceleration: Co-Design and Deployment in the Wild. In *ASPLOS*.
[41] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated Model-less Inference Serving. In *ATC*.
[42] Francisco Romero, Mark Zhao, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. Llama: A Heterogeneous & Serverless Framework for Auto-Tuning Video Analytics Pipelines. In *SoCC*.
[43] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation. In *OSDI*.
[44] Prahlad Venkatapuram, Zhao Wang, and Chandra Mallipedi. 2020. Custom Silicon at Facebook: A Datacenter Infrastructure Perspective on Video Transcoding and Machine Learning. In *IEDM*.
[45] Midhul Vuppalapati, Justin Miron, Rachit Agarwal, Dan Truong, Ashish Motivala, and Thierry Cruanes. 2020. Building An Elastic Query Engine on Disaggregated Storage. In *NSDI*.
[46] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. 2020. AnalyticDB-V: A Hybrid Analytical Engine towards Query Fusion for Structured and Unstructured Data. *Proc. VLDB Endow.* (2020).
[47] Tiantu Xu, Luis Materon Botelho, and Felix Xiaozhu Lin. 2019. VStore: A Data Store for Analytics on Large Videos. In *EuroSys*. ACM, 16.
[48] Fangjin Yang, Eric Tschetter, Xavier Léauté, Nelson Ray, Gian Merlino, and Deep Ganguli. 2014. Druid: A Real-Time Analytical Data Store. In *SIGMOD*.
[49] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *NSDI*, Vol. 9. 1.