V2V: Efficiently Synthesizing Video Results for Video Queries

Dominik Winecki
Computer Science & Engineering
The Ohio State University
Columbus, USA
winecki.1@osu.edu

Arnab Nandi
Computer Science & Engineering
The Ohio State University
Columbus, USA
nandi.9@osu.edu

***Abstract*—Querying video data has become increasingly popular and useful. Video queries can be complex, ranging from retrieval tasks *("find me the top videos that have...")*, to analytics *("how many videos contained object X per day?")*, to excerpting tasks *("highlight and zoom into scenes with object X near object Y")*, or combinations thereof. Results for video queries are still typically shown as either relational data or a primitive collection of clickable thumbnails on a web page. Presenting query results in this form is an impedance mismatch with the video medium: they are cumbersome to skim through and are in a different modality and information density compared to the source data. We describe *V2V*, a system to efficiently synthesize *video results for video queries*. V2V returns a fully-edited video, allowing the user to consume results in the same manner as the source videos. A key challenge is that synthesizing video results from a collection of videos is computationally intensive, especially within interactive query response times. To address this, V2V features a grammar to express video transformations in a declarative manner and a heuristic optimizer that improves the efficiency of V2V processing in a manner similar to how databases execute relational queries. Experiments show that our V2V optimizer enables video synthesis to run 3× faster.**

***Index Terms*—multimedia databases, video result synthesis, declarative video editing**

I. Introduction

Video data has experienced a remarkable surge, impacting many aspects of our daily lives. The widespread adoption of smartphones, social media platforms, and other digital devices has contributed to the exponential growth in video content creation and consumption. As a result, the need to efficiently query and analyze video data has become increasingly vital.

Video Database Management Systems (VDBMSs), Computer Vision models, and hybrid Large Language/Vision models have made tremendous progress toward machine understanding of video, largely solving video-to-relation querying. However, relational data is a poor human interface for multimedia data. Instead, returning a single *video* can provide a more intuitive and digestible result. While VDBMSs support extracting clips and others have hard-coded visualizations for specific tasks, such as drawing bounding boxes, our survey found gaps in the expressiveness of result videos and no research on efficiently materializing them.

Video plays an ever-expanding role in our consumption of data, and through interactive data systems, it is also becoming a larger part of our interaction with data. The evolution of interfaces reflects this: desktop systems have a wide variety of data models, but video is a clear standout on mobile devices, where playing a single video as opposed to a set of video results yields a better interface, as shown by the proliferation of short-form video. As such, video results enable first-class mobile device data interaction. Beyond mobile devices, Augmented and Virtual Reality systems also have first-class video results, especially as little data is natively volumetric. In this future class of metaverse devices, we have more opportunities to use multimedia queries, especially over media captured by the device itself, in addition to the need to summarize the videos effectively in response to queries.

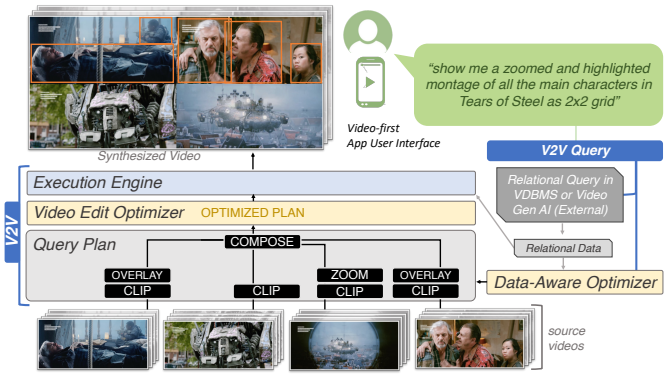

Fig. 1. V2V System Architecture

Motivating Example: Consider asking "Show me all the times zebras exhibited social behavior and overlay their IDs and the behavior type". We assume there are existing tables that contain the needed information. The result must combine potentially thousands of source videos, zoom into the correct spot, and overlay informational text over the relevant zebras in the frame. Today, such a video result can only be created by handwriting code for this specific task. Additionally, the resulting video could be multiple hours long, which would take long to process. Through database-style optimizations described in this paper and on-demand streaming, V2V enables a VDBMS to execute such a query and to begin playback within seconds.

Video Synthesis: The creation of new videos from existing video and data sources, which we term *video synthesis*, is a task that has emerged as a crucial aspect of managing and processing the ever-expanding pool of video content. The objective is to create coherent and meaningful video sequences that cater to specific user requirements. We focus on video-to-video queries, which return a video created from existing videos. These go beyond simple clip operations; for example, *"show me the event from multiple cameras as 2×2 grid with object overlays"* includes *composition* and *data overlay*. Such synthesis applies to a diverse range of applications, impacting several use cases that are both practical and relevant. Since this task joins with external data while editing, this is a video synthesis task instead of a video editing task.

Use Cases: Video editing is already widely used, often done manually, and is the primary method for video summarization. Video-first social media platforms such as TikTok, Instagram Reels, and YouTube Shorts have made short-format videos a mainstay, necessitating the condensation of content into fast-paced, information-dense videos. In sports and live streaming (e.g., Twitch), highlight reels are popular, condensing hours of gameplay into sequences showcasing a game's most pivotal and entertaining moments. In cinema, filmmakers and fans produce trailers or "supercuts," seamlessly weaving together themes and iconic quotes from movies into short, fast-paced clips. Furthermore, video editing is valuable for summarizing presentations and meetings in the workplace.

Beyond video editing, video synthesis is necessary due to the growth of video analytics in industry and scientific areas such as modern wildlife conservation and ecology research [1]. Such work relies heavily on computer vision over images and video footage, critical to our planet's sustainable future. Scientific applications of video analysis range from understanding animal behavior such as foraging [2] or hunting events [1], to tracking animals using animal biometrics [3]. Here, even a short video collection activity (e.g., drone-based footage of giraffes and zebras in Kenya [4], one of this paper's evaluation datasets) can yield a large amount of data (1.1TB across 377 videos) that can be overwhelming for researchers to analyze. Empowering these researchers with an effective and efficient platform to explore video data is extremely valuable.

Another potential application of video synthesis is in the nascent and rapidly growing space of generative AI for video storytelling [5]. Instead of receiving videos from a VDBMS, videos could be *generated* from prompts: State-of-the-art text-to-video generation projects such as Emu [6], RunwayML Gen-2, and stability.ai are now capable of producing impressive multi-second clips from textual prompts, combining them with impressive multi-task image editing models. Combining such clips with existing libraries of videos to synthesize longer, human-watchable narratives is a useful and desirable task, and will also require a video synthesis engine such as V2V.

The importance of low response times: For many of the use cases above, we expect this synthesis to occur in an ad-hoc manner to summarize and present the results of a query. This could be the result of conventional video queries issued by an analyst, from personalizing videos for a specific user, or from content-based queries by end-users (e.g., *tap and hold on a car in an Instagram Reel to see a video montage of that car*). Low response times are critical to a positive user experience.

The case for a result synthesis engine: The area of video database management systems (VDBMS) has seen notable interest in the recent past, including systems such as EVA [7], VOCAL [2], VIVA [8], and BlazeIt [9]. For most current systems, video results to queries are limited to basic clips of individual videos. Instead, we envision the user asking an analytical V2V query over their video collection and returning a *single* easy-to-watch video for their analysis. This video is optimized for easy consumption: relevant portions are zoomed in, objects are annotated, and multiple clips are played side-by-side. After watching this video, they can ask further questions about the existing video or continue their analysis by looking at additional data. While some systems provide post-processing options as Python scripts, our work observes that the video synthesis step can also be declaratively specified, and can take advantage of several low-level optimizations. Another benefit to having a single video as a final output is that it allows for a closed query algebra, enabling users to express complex compound query operations.

Hence, we consider V2V a complementary and pluggable engine for existing VDBMSes, managing the result synthesis step of the video querying pipeline. Conversely, V2V can run independently without a VDBMS for a variety of use cases, including generative video storytelling.

Challenges: Synthesizing video results presents several non-trivial computational challenges that we address in V2V. First, it inherits all of the difficulties with video editing: video processing involves sifting through raw raster data streams, which can be massive. Since we expect swift response times as video queries are expected to be run in ad-hoc interactive query sessions, making editing efficient is paramount.

The heterogeneity of video codecs, file formats, and processing tools triggers a complex interoperability problem for video data pipelines. Additionally, we need to coalesce different paradigms in video editing. For example, highlighting objects considers the contents of each frame, but splicing together videos considers entire video timelines. A stream/file block-centric approach (e.g., used by FFmpeg) is better suited for bulk operations; frameworks such as OpenCV are more aligned with a per-frame image-centric model and are better suited for content-focused operations. Hence, abstractions, query grammars, optimizations, and implementations must be carefully crafted to capture semantics.

In addition to video editing difficulties, video synthesis incorporates joining video edits with relational data. This requires data access interfaces, methods of defining how to incorporate data in video transformations, and an execution engine capable of fetching the needed data while performing an edit. The system should also use the specific data values to further optimize plans.

Contributions: Our contributions in this work are as follows:
- A DSL for representing video synthesis tasks.
- An OLAP-style optimizer which optimize video synthesis tasks similarly to relational queries.
- A video editing specification rewrite system to enable data-aware optimizations.

II. Related Work

VDBMS Which Return Videos: Many existing VDBMS systems, across research, open source, and industry, can display videos or frames to the user [10]–[15], most of which only as a link to the video source or a basic clip. However, how such result videos are created is, in all papers we have surveyed, unspecified. For example, VIVA [8] allows basic clip selection, but how these clips are generated is left open-ended. We have also found no shortage of supporting scripts which perform video tasks that VDBMSs are unable to. For example, EVA [7] provides post-processing scripts to generate annotated videos from tasks such as object detection, and Spatialyze [16] does the same in a utility function.

VDBMS Architectures and Optimizations: VDBMSs can use video-specific optimizations in MapReduce architectures [17], serverless pipelines [18], [19], or on edge platforms for real-time stream analysis [20]. SEIDEN [21] explored using techniques such as query-agnostic indexing, sampling additional frames, and leveraging temporal continuity. Similarly, TASTI [22] uses semantic indexing.

VDBMSs tend to push their storage needs into a dedicated storage layer and, similarly to relational queries, pushing operators into the storage layer yields significant speedups [23]. Additionally, since video is often accessed through external tools like FFmpeg, pushing operations into those tools also improves performance [24]. TASM [25] tiles videos to ensure they only decode regions of video needed by a query. Other systems, such as VStore [26] only use temporal sharding but stores cached copies of commonly-used video artifacts.

Querying Videos: Issuing queries on video data usually takes the form of a DSL [11], [27], [28] or a graphical system [29]. Most of these DSLs are derivatives of SQL [9], [30], [31] or Prolog [15], [32]. In addition to these formal representations of queries, natural language queries [33], [34] and voice queries [35] have both been explored over video. VOCAL [2] enables diverse compositional analytical queries without requiring a semantic model. Additionally, emerging AI-centric vector/tensor databases support unstructured images, video, and audio [36]. As discussed in the previous section, results from these VDBMSs can hugely benefit from a complementary result synthesis engine such as V2V.

In addition to querying content, both filming and editing techniques, such as *cuts*, *zooms*, and *panning*, can be queried as temporal events [37], [38].

Declarative queries against video data allow for transparently applying advanced video-data-specific optimizations to query plans [7], [9], [39]–[41] indexing [42], or both [43]. In addition to these general-purpose video optimizations there have been substantial undertakings to optimize data systems *specific* to certain video analytic tasks, usually by hard-coding the plans for their common tasks. Traffic analysis [44] is the most common, followed by tracking individuals from security cameras [45].

Video Editing DSLs: DSLs for expressing video edits have been explored previously [46], [47], each focusing on different scopes. Super 8 [46] is a creativity-focused language for creating edits comprising of segments of a certain length interlaced with transitions and text slides. Similarly, editly [47] performs similar tasks through its command-line interface while supporting more complex transformations when using a more expressive JSON-based editing specification. It limits its video transformations to a few hand-implemented ones and applies them at a clip-level. Additionally, the optimizations are limited to a "fast" option which sets a low resolution and frame rate, which makes it the only declarative video editing tool we have found which supports any optimization at all.

A declarative approach amenable to further automated optimization has been unexplored, as have DSLs supporting joining against data, motivating the need for V2V.

Conventional Editing Techniques: Supercuts [48] are popular style of montages consisting of short clips. Systems to stitch together supercuts automatically have been explored [49].

Video Processing: Scanner [50] uses a distributed graph data-flow processing approach to video processing. While bulk video analysis and the data-flow processing are unsuitable to low-latency video editing on varying specifications, Scanner's decoder uses keyframe indexes for partial group-of-pictures (GOP) decodes similar to V2V. LosslessCut [51] and avcut [52] are two open-source projects which use knowledge of GOPs/keyframes to speedup up clipping video without a full decode/encode.

ML Video Synthesis: The contributions of our work are orthogonal and complementary to generative machine learning-based video techniques presented in [53], [54]; our interpretation of the word "synthesis" represents the combination of existing videos, while the aforementioned work focuses on human-guided generation of videos.

VVS [55] is a content-based video retrieval system which removes irrelevant details via a ML network. While VSS is adjacent to the information retrieval tasks we target, we focus on quickly editing videos while VVS focuses on identifying what details should be removed from videos via editing.

III. Declarative Video Editing

At its core, video synthesis is built upon video editing. We propose a declarative video editing (DVE) system which serves as the core of our video synthesis system. We developed a custom DVE system as the few existing systems were not sufficiently expressive for our needs. Specifically, they only allowed for applying a transformation across an entire time span, defined times in imprecise floating-point seconds, and could not combine multiple input videos.

Note that our DVE system is not specific to the video synthesis use case. It is generalized with video synthesis

abilities implemented on top, so further developments in DVE systems can also benefit video synthesis. A video editing task is expressed as a ***spec***, which takes videos as input and returns a single video as output.

A. Data Model

A **video** is an **array** of frames at specific times. Each array is indexed by a timestamp, represented as a rational number. This is the standard representation in multimedia since many frame rates (e.g., 29.97, 30 FPS) can not be exactly represented as finite-length decimal. A **frame** is our model's smallest unit of information. We consider a frame arbitrary data of a specific type; for example, a `1920x1080` frame with `yuv420p`-encoded `BT.709` color. The flexibility of the data allows non-standard formats, like 3D video [56], to be represented while ensuring type correctness.

B. Specification Model

A DVE spec represents transformations over input videos, which are a series of transformations over frames. Since the output of a spec is a video, we construct an array of frames where each frame has specific transformations. An example DVE spec is as follows:

```
Spec = <TimeDomain, Render,
  videos: {"vid1": "video1.mp4", ...}>
```

where `TimeDomain` defines a 10-minute video at 30 FPS:

```
TimeDomain = Range(0, 600, 1/30)
```

We use `Range` as shorthand for a set of evenly spaced rationals over an interval. `Render` is defined to provide transformations needed on each frame when given the frame's time. Here we concatenate two 5-minute videos, with the second as our prior example montage:

```
Render(t) = match t {
    t in Range(0, 300, 1/30) => vid1[t],
    t in Range(300, 600, 1/30) =>
        Grid(
            vid1[t + 13463/30],
            Overlay(vid2[t], "overlay.png"),
            Zoom(vid3[t], 10.0),
            vid4[t + 9952/30]
        ),
}
```

Transformations are modeled as functions. Given frame x and transformation Transform(Frame) $\rightarrow$ Frame we can apply the transformation with Transform(x), which has type *Frame*. A transformation can include multiple frames and data, for example:

- Zoom(Frame, percent : Number) $\rightarrow$ Frame
- Overlay(Frame, image_path : String) $\rightarrow$ Frame
- BoundingBox(Frame, List⟨BoxCoord⟩) $\rightarrow$ Frame
- Grid(Frame, Frame, Frame, Frame) $\rightarrow$ Frame

These frame transformations can be created by the V2V module or in user-defined functions (UDFs). Each frame can be indexed as `video[t]` where `t` is the time. Such a definition has two main benefits: First, it is an unambiguous method of describing video transformations in a VDBMS, similar to how relational queries are unambiguous data transformations. And second, it allows static property checks via typing. For this example spec, our system would identify that `vid1` must be a superset of `Range(0, 300, 1/30)`. The spec is correct if each dependency is a subset of the ranges available in the source videos.

C. Operators

Our execution engine consists of three core operators, which enable running specs with a multitude of common editing operations:

- `Concat`: Splice together multiple segments
- `Clip`: Extract a specific time segment
- `Filter`: Zoom, crop, stabilize, animated transitions, highlight an object, overlay text or graphics, color grading, blur/sharpen, edge detection, denoise, background replacement

More transformations can be added through UDFs. A transformation in the spec takes some combination of frames, data, and time (e.g., for an animated transition) and returns a single transformed frame. A sequence of these forms a `Filter` operator. `Concat` and `Clip` are temporal, as they control which operations should occur on which frames at a specific time. Every spec comprises concatenations of frame transforms on ranges of videos selected by clips. We form an unoptimized logical plan by mapping our declarative definition to these operators where match operators create `Concat`s, function calls create `Filter`s, and the indexing of videos with time results in `Clip`s.

D. Declarative Video Editing Optimizations

We implement several heuristic rewrite-based optimizations. These include conventional general-purpose optimization techniques, such as temporal sharding and operator merging, in addition to domain-specific optimizations, such as *stream copying* and *smart cuts*.

a) Stream Copying: When performing splices or simple clips without transformations, we can avoid re-encoding the source. This is simple in splices: multiple compatible video streams in the same codec can be concatenated into a single stream [57]. When clipping, a stream copy is only frame-exact if it happens to land on keyframe boundaries (often called *group of pictures* or GOP), which is rare. Instead, our DVE uses an alternative: *smart cuts*, a new approach experimentally supported by open-source editors such as LosslessCut [51]. A smart cut finds the first keyframe in the clipped range and stream copies until the last. Any frames preceding the first keyframe are re-encoded with the result spliced onto the start; the same is done to the end of the clip. This allows for the exact clipping of arbitrarily large clips while only re-encoding at most two GOPs.

b) Example Optimized Plans: Consider a spec that performs a simple clip spliced with a 2x2 grid spliced with a simple filter (in our evaluation, these are specs $Q1$, $Q3$, and $Q4$, respectively). The top of Fig. 2 shows the unoptimized plan for this spec. In this diagram, we denote stream-copy operators as diamond-shaped grey nodes. An optimized plan

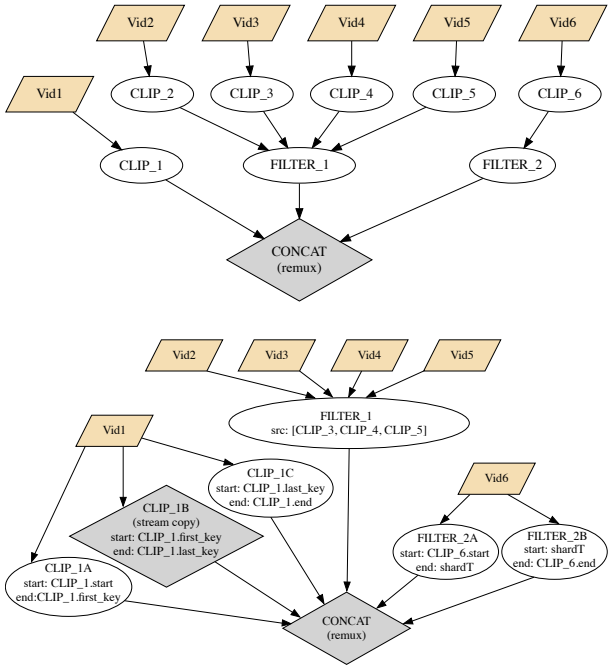

Fig. 2. Unoptimized (top) and Optimized (bottom) Plans

is shown below. We have applied a smart cut to the clip where the bulk of the clip is copied from the input to the output without re-encoding. On the 2x2 grid, we have pulled the clip into the filter operator. This merges the clip operation into the filter, avoiding an unnecessary encode/decode pair. We split the last filter operator into two parts that run in parallel.

IV. The V2V System

A. System Architecture and Implementation

We propose a V2V as a video synthesis expansion of the declarative video editing system detailed in Section III. V2V declaratively defines a video edit with the additional capability of allowing relational data to be integrated with the edit. It does so by adding the additional step of data-aware rewriting.

Data-Aware Rewriter: The data-aware rewriter takes a V2V specification and rewrites the spec to incorporate data-aware optimizations. The result of the data-aware rewriter is equivalent to the input spec on the specific data it references.

Video Edit Optimizer: The video optimizer takes the previously rewritten spec and turns it into a plan for the execution engine. In the process, we perform type-checking to ensure that all referenced videos, arrays, and members thereof are valid. Following this, the optimizer makes multiple passes to apply optimizations, as listed in Sec. III-D. The resulting plan may pass data to the filters like it passes frames, but the video optimizer does not consider their values in the optimizer.

Execution Engine: The V2V execution engine runs the completed plan. Our implemented execution engine runs a plan AST and uses FFmpeg for all operators. We use the dependency graph to execute operators in parallel as an additional optimization at runtime.

Integrating into VDBMSs: We envision V2V as a pluggable module that provides video synthesis functions for existing VDBMSs. A user can issue a query in natural language, and the VDBMS can use an LLM to write SQL queries over cached model results *and* the V2V spec, turning that relational data into a synthesized video. The VDBMS first runs the query, which yields a relation detailing what videos are to be used and then this is transformed into a V2V spec. The V2V system then transforms the spec into a plan, optimizes it, and executes it to synthesize the output video, optionally taking advantage of any storage optimizations the VDBMS provides. The VDBMS then passes the synthesized video to the client.

B. Data in Specs

Video synthesis must enable joining relational data with video data. Our DVE engine supports data as additional constant parameters. For example, the Overlay operation takes the path of an image to overlay over a frame. We can expand this syntax to support arbitrary data expressions as well. Since videos are already represented as arrays, we introduce *data arrays*. For example, if we have bounding boxes in a JSON file, we could write a V2V spec as such:

```
TimeDomain = Range(0, 300, 1/30)
Render(t) = BoundingBox(vid1[t], vid1_bb[t])

Spec = <TimeDomain, Render,
  videos: {"vid1": "video1.mp4", ...},
  data_arrays: {"vid1_bb": "annot1.json"}>
```

Similarly, we may support arbitrary SQL as well, given that it takes the form of a tuple of a rational timestamp and a scalar element:

```
SELECT timestamp, frame_objects
  FROM video_objects
  WHERE video = ... AND model = "yolov5m";
```

Using SQL to define data joined with videos allows intelligent data materialization. The data queries may be materialized in portions by bounding the time, which allows for fine-grained control between storage and compute.

C. Data-Dependent Rewrites

While supporting joining data within specs is the minimum needed to enable video synthesis, the DVE engine can use the joined data to further optimize plans. We term this *data-dependent rewriting*, which allows spec rewrites to find more optimal equivalent specs.

We demonstrate these rewrites by considering two operators: IfThenElse and BoundingBox. IfThenElse takes parameters of type (bool, Frame, Frame). However, naive support for joining data in specs does not allow efficient execution of such a function. This operator is superfluous in conventional DVE, which is not aware of data, as the conditional variable would be constant. Our DVE optimizer cannot look at data directly while optimizing, and adding such support to the underlying optimizer would be a significant task. As such, the optimizer

would materialize both the left and right frame references. Then, only at runtime would the operator select which frame to use. This is sub-optimal.

A more subtle optimization occurs around the BoundingBox operator with parameters (Frame, List⟨BoxCoord⟩). When a frame has objects, the BoundingBox operator draws rectangles over their region, transforming the frame. However, many frames may not contain any objects. In those cases, the BoundingBox operator is equivalent to the identity transformation. This is important as DVE engines can use stream copies on unmodified frames, the fastest class of video edits operating near the speed of a memory copy. Thus, when a video has no objects over the length of a group of pictures, it can be stream-copied.

It is possible to hard-code such optimizations, but DVE engines are complex, so we propose another solution to achieve data-aware optimizations: data-dependent rewrites.

We use a two-pass execution method: the first is data-only, and the second is the full execution. The first data-only pass applies rewrites to the spec based on the data referenced by the spec. Each operator is associated with a new *data-dependent equivalence* function, denoted as f_{dde}. This function only takes non-frame 'relational data' parameters and returns an equivalent expression. For example, consider array a = [3, 6, 8] and the following spec:

```
TimeDomain = {0, 1, 2}
Render(t) =
  IfThenElse(a[t] < 5, vid1[t], vid2[t])
```

Since IfThenElse has parameters (bool, Frame, Frame), then $\text{IfThenElse}_{\text{dde}}$ has the same parameters but may only perform computation on the boolean parameter and not the frames which are included only as symbolic placeholders:

$$\text{IfThenElse}_{\text{dde}}(c, x, y) = \begin{cases} x & \text{if } c \\ y & \text{if } \neg c \end{cases}$$

The data-dependent equivalence function is applied across the entire spec. This yields the data-dependent DVE spec:

```
TimeDomain = {0, 1, 2}
Render(t) = match t {
    t in {0} => vid1[t],
    t in {1, 2} => vid2[t],
}
```

Similarly, the BoundingBox is the identity transform if and only if it has no objects on that frame:

$$\text{BoundingBox}_{\text{dde}}(x, b) = \begin{cases} x & \text{if } |b| = 0 \\ \text{BoundingBox}(x, b) & \text{if } |b| > 0 \end{cases}$$

By running each spec twice, first with data-dependent equivalence operators and second with full frame operators, we can create specs that are easier to optimize efficiently without needing data-aware optimizations in the declarative video editing optimizer.

D. Implementation

The V2V system is implemented in the Rust language, and we use FFmpeg as the underlying execution engine. This takes full advantage of the parallelism support offered by both. Specs and plans are represented using typed abstract syntax trees (ASTs), taking advantage of Rust's algebraic data type system in our library API, and our executable binary reads serialized JSON specs.

V. Evaluation

Datasets: We evaluate the V2V system over two video collections, *ToS* and *KABR* datasets. The first is clips from *Tears of Steel* (ToS) from Blender Open Movies, a popular video benchmarks film [58]. We preprocessed the film to overlay frame information to verify each operation was frame-exact; the video is 734 seconds at 3840×1714 resolution at 24fps, and the bitrate is 18.8Mb/s H.264.

For our second dataset, we use raw footage from the KABR effort [4], collected by researchers using drones from a site visit to Kenya. We chose this dataset as it will resemble many large scientific datasets. We used $4\times$ 291-second videos at 3840×2160 resolution encoded at $\sim$ 103Mb/s H.264.

Benchmarks and Environment: To evaluate performance, we use a benchmark representing common synthesis tasks:

1) Clip a segment of video
2) Clip 4 segments of video and splice them together
3) Clip 4 segments of video and play them simultaneously in different quadrants of the output video
4) Clip a segment of video and apply pixel-wise filter operation (we used a Gaussian blur)
5) Clip a segment of video and draw object bounding boxes and class annotations

We use these tasks as our base and evaluate each with 5-second and 1-minute input segments. In our tests, we term these queries with 5-second inputs as $Q1 - Q5$ and $Q6 - Q10$ for the 1-minute inputs. We set the output format to be 1280x720 video in H.264 with the `ultrafast` encoding preset. We used a 2×12-core (48 HT vCPUs) Intel Xeon Gold 6126 @ 2.6 GHz system with 384GB DDR4 RAM running Ubuntu 18.04 LTS and FFmpeg 6.0. Care was taken to flush caches and buffers between invocations, and averages of 5 runs were measured after discarding an initial run.

A. V2V Synthesis Performance

The primary design goal of the V2V system is to enable video synthesis in interactive settings. In this light, we evaluated the end-to-end synthesis performance of our V2V system (planner, optimizer, and execution engine), running the benchmark queries against each of our 2 datasets. We ran the unoptimized and optimized plans and measured the average execution time.

As shown in Figs. 3 and 4, on average, the optimized queries ran $3.44\times$ faster for the ToS dataset, and $5.07\times$ faster for the KABR dataset than their unoptimized counterparts. Of

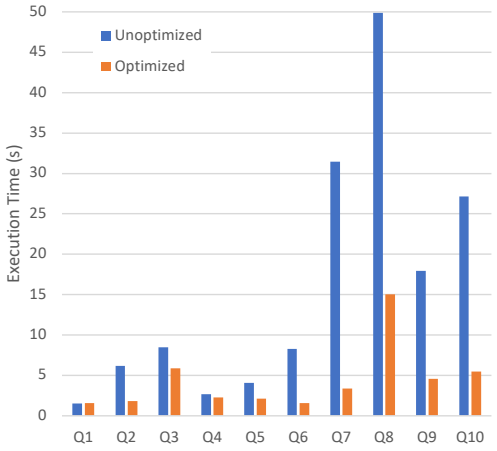

Fig. 3. Spec Execution Times - ToS Dataset

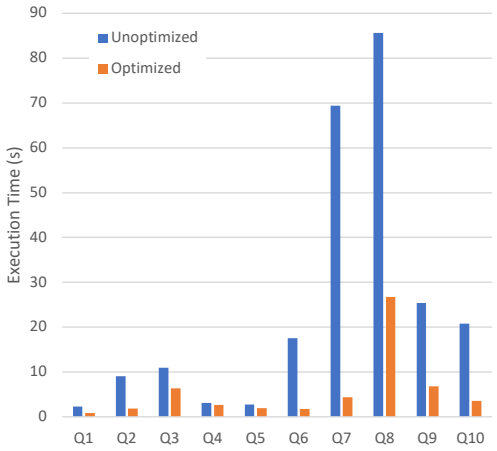

Fig. 4. Spec Execution Times - KABR Dataset

note are Q6 and Q7 on the KABR dataset, which result in 736MB 1-minute and 2.9GB 4-minute videos, now executing in 1.75 and 4.34 seconds, respectively. The speedups through our V2V optimizer enable these queries to be usable in interactive settings. We observed subtle but interesting performance insights in our evaluations. The optimized and unoptimized plans for Q1 on ToS were identical, as there were insufficient keyframes over the clipped region to apply a smart cut. Q1 on KABR did perform a smart cut since it had keyframes every second. On all other queries, we found substantial performance improvements on both datasets.

We also compared the queries that joined with data against the equivalent Python + OpenCV equivalents. These results are shown in Fig. 5. The encoding/decoding for the OpenCV scripts also used FFmpeg, so the codec overhead should be identical. We found that the ToS dataset has objects on nearly every frame, whereas the KABR dataset only occasionally has a zebra caught by the object detector. This translated to a speedup on the KABR dataset from the data-aware rewrites, which removed most bounding box filters and allowed stream-copies on some segments.

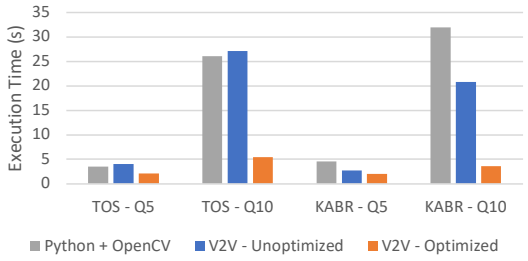

Fig. 5. Video Synthesis Data Join Comparison

VI. Conclusion

This paper articulates the need for a video result synthesis engine in VDBMSs and presents an efficient and effective implementation. Video synthesis combines editing a video from one or more sources while joining with relational data. We envision a video synthesis engine as an embedded module within a VDBMS that creates result videos. This enables VDBMSs to take complex queries and quickly return a single expressive video result.

Video result synthesis allows for rapid iteration in video analytics pipelines, which is especially important across diverse scientific fields. However, while video synthesis is useful, it can be a computationally challenging task with tremendous scope for optimization. To address this, we expand a declarative video editing grammar to enable synthesis specs that reference data and for the optimizer to create data-aware optimizations. Our experiments show that V2V optimizations enable synthesis to run $3.4\times$ faster on the ToS dataset and $5.1\times$ on the KABR dataset. Certain queries, such as Q6 applied to the KABR dataset, demonstrate a substantial improvement in execution speed, approximately $16\times$, reducing the execution time from 69 seconds to 4.3 seconds. This enhancement transitions the performance from being prohibitively slow to feasible for interactive environments. On tasks that include joining with data (e.g., Q5 & Q10), we average a $4.4\times$ speedup against the equivalent Python+OpenCV script.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Award No. 1910356.

References

[1] N. Haering, R. J. Qian, and M. I. Sezan, "A semantic event-detection approach to detecting hunts in wildlife video," *IEEE TCSVT*, 2000.

[2] M. Daum, E. Zhang, D. He, M. Balazinska, B. Haynes, R. Krishna, A. Craig, and A. J. Wirsing, "Vocal: Video organization and interactive compositional analytics," *CIDR*, 2022.

[3] M. Lahiri, , D. Rubenstein, T. Berger-Wolf *et al.*, "Biometric animal databases from field photographs: identification of individual zebra in the wild," in *ICMR*, 2011.

[4] M. Kholiavchenko, T. Berger-Wolf, D. Rubenstein, C. Stewart *et al.*, "KABR: In-Situ Dataset for Kenyan Animal Behavior Recognition from Drone Videos," *Preprint*, 2023. [Online]. Available: https://dirtmaxim.github.io/kabr

[5] T. Turchi, S. Carta, L. Ambrosini, and A. Malizia, "Human-ai co-creation: Evaluating the impact of large-scale text-to-image generative models on the creative process," in *International Symposium on End User Development*. Springer, 2023, pp. 35–51.

[6] S. Sheynin, A. Polyak, U. Singer, Y. Kirstain, A. Zohar, O. Ashual, D. Parikh, and Y. Taigman, "Emu edit: Precise image editing via recognition and generation tasks," *arXiv preprint arXiv:2311.10089*, 2023.

[7] G. Kakkar, J. Cao, P. Chunduri, J. Arulraj *et al.*, "Eva: An end-to-end exploratory video analytics system," in *DEEM*, 2023.

[8] D. Kang, F. Romero, P. Bailis, C. Kozyrakis, and M. Zaharia, "VIVA: an end-to-end system for interactive video analytics," in *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. CIDR, 2022.

[9] D. Kang, P. Bailis, and M. Zaharia, "Blazeit: Optimizing declarative aggregation and limit queries for neural network-based video analytics," 2019.

[10] E. Hwang and V. Subrahmanian, "Querying video libraries," *journal of visual communication and image representation*, vol. 7, no. 1, 1996.

[11] U. Arslan, "A semantic data model and query language for video databases," Ph.D. dissertation, Bilkent Universitesi (Turkey), 2002.

[12] T. Catarci, M. Donderler, E. Saykol, O. Ulusoy, and U. Gudukbay, "Bilvideo: A video database management system," *IEEE MultiMedia*, vol. 10, no. 1, 2003.

[13] D. Kang, P. Bailis, and M. Zaharia, "Challenges and opportunities in dnn-based video analytics: the blazeit video query engine." *CIDR*, 2019.

[14] P. Zhang, C. Yuan, K. Liu *et al.*, "Fast video clip retrieval method via language query," *ADMA*, 2019.

[15] E. Zhang, M. Daum, D. He, M. Balazinska, B. Haynes, and R. Krishna, "Equi-vocal: Synthesizing queries for compositional video events from limited user interactions," *arXiv:2301.00929*, 2023.

[16] C. Kittivorawong, Y. Ge, Y. Helal, and A. Cheung, "Spatialyze: A geospatial video analytics system with spatial-aware optimizations," 2023.

[17] C. Ryu, D. Lee, M. Jang, C. Kim, and E. Seo, "Extensible video processing framework in apache hadoop," *CCTS*, 2013.

[18] L. Ao, L. Izhikevich, G. M. Voelker, and G. Porter, "Sprocket: A serverless video processing framework," *SoCC*, 2018.

[19] M. Zhang *et al.*, "Charmseeker: Automated pipeline configuration for serverless video processing," *IEEE/ACM Transactions*, Dec 2022.

[20] M. Ali, A. Anjum, O. Rana, A. R. Zamani, D. Balouek-Thomert, and M. Parashar, "Res: Real-time video stream analytics using edge enhanced clouds," *IEEE Transactions on Cloud Computing*, vol. 10, no. 2, 2022.

[21] J. Bang, G. Kakkar, P. Chunduri, S. Mitra, and J. Arulraj, "Seiden: Revisiting query processing in video database systems," in *VLDB*, 2023.

[22] D. Kang, J. Guibas, P. D. Bailis, T. Hashimoto, and M. Zaharia, "TASTI: Semantic Indexes for Machine Learning-based Queries over Unstructured Data," *SIGMOD*, 2022.

[23] B. Haynes, M. Daum, D. He, A. Mazumdar, M. Balazinska, A. Cheung, and L. Ceze, "Vss: A storage system for video analytics," *SIGMOD*, 2021.

[24] X. Wu, P. Qu, S. Wang, L. Xie, and J. Dong, "Extend the ffmpeg framework to analyze media content," *CoRR*, vol. abs/2103.03539, 2021. [Online]. Available: https://arxiv.org/abs/2103.03539

[25] M. Daum, B. Haynes, D. He, A. Mazumdar, and M. Balazinska, "TASM: A Tile-Based Storage Manager for Video Analytics," *ICDE*, 2021.

[26] T. Xu, L. M. Botelho, and F. Lin, "VStore: A Data Store for Analytics on Large Videos," *EuroSys*, pp. 1–17, 2019.

[27] H. Arisawa, T. Tomii, and K. Salev, "Design of multimedia database and a query language for video image data," *IEEE MCS*, 1996.

[28] M.-S. Hacid, C. Decleir, and J. Kouloumdjian, "A database approach for modeling and querying video data," *TKDE*, 2000.

[29] S. Hibino and E. A. Rundensteiner, "A visual query language for identifying temporal trends in video data," *MMDMS*, 1995.

[30] S. Jo and I. Trummer, "Thalamusdb: Answering complex sql queries with natural language predicates on multi-modal data," in *SIGMOD*, 2023.

[31] B. Haynes and M. Daum, "Visualworlddb: A dbms for the visual world. brandon haynes, maureen daum, amrita mazumdar, magdalena balazinska, alvin cheung, and luis ceze," *CIDR*, 2020.

[32] M. Dönderler *et al.*, "Bilvideo: Design and implementation of a video database management system," *Multimedia Tools and Applications*, vol. 27, no. 1, 2005.

[33] G. Erozel, N. K. Cicekli, and I. Cicekli, "Natural language querying for video databases," *Information Sciences*, vol. 178, no. 12, 2008.

[34] F. Romero, C. Winston, J. Hauswald, M. Zaharia, and C. Kozyrakis, "Zelda: Video analytics using vision-language models," *arXiv:2305.03785*, 2023.

[35] T. Bhowmik, A. Rai, S. Pandey, P. Boddu, N. Patel, V. S, and V. Manchala, "A novel approach towards voice-based video content search," in *I2CT*, April 2021.

[36] A. Gandhi, Y. Asada, V. Fu, A. Gemawat, L. Zhang, R. Sen, C. Curino *et al.*, "The tensor data platform: Towards an ai-centric database system," *arXiv:2211.02753*, 2022.

[37] J. Assfalg, A. Del Bimbo, and M. Hirakawa, "A mosaic-based query language for video databases," in *IEEE VL*, 2000.

[38] T. C. Kuo and A. L. Chen, "Content-based query processing for video databases," *IEEE Transactions*, vol. 2, no. 1, 2000.

[39] J. Cao, K. Sarkar, R. Hadidi, J. Arulraj, and H. Kim, "Figo: Fine-grained query optimization in video analytics," *SIGMOD*, 2022.

[40] F. Romero, J. Hauswald, A. Partap, D. Kang, M. Zaharia, and C. Kozyrakis, "Optimizing video analytics with declarative model relationships," *VLDB*, 2022.

[41] Z. Xu, G. Kakkar, J. Arulraj, and U. Ramachandran, "Eva: A symbolic approach to accelerating exploratory video analytics with materialized views," *SIGMOD*, 2022.

[42] D. Kang, J. Guibas, P. D. Bailis, T. B. Hashimoto, and M. A. Zaharia, "Task-agnostic indexes for deep learning-based queries over unstructured data," *ArXiv*, vol. abs/2009.04540, 2020.

[43] Y. Chen, X. Yu, and N. Koudas, "Ranked window query retrieval over video repositories," *ICDE*, 2022.

[44] G. H. Apostolo, P. Bauszat, V. Nigade, H. E. Bal, and L. Wang, "Live video analytics as a service," in *European Workshop on ML and Systems*, 2022.

[45] A. Sinha *et al.*, "Content based person retrieval from video using forward backward frame check algorithm," *ICoAC*, 2015.

[46] L. Andersen, S. Chang, and M. Felleisen, "Super 8 languages for making movies (functional pearl)," *PACMPL*, vol. 1, no. ICFP, Aug. 2017.

[47] (2023) editly. [Online]. Available: https://github.com/mifi/editly

[48] M. Tohline, "A supercut of supercuts: aesthetics, histories, databases," *Open Screens*, 2021.

[49] D. C. Heras, "Creanalytics: Automating the supercut as a form of critical technical practice," *Convergence*, 2023.

[50] A. Poms, W. Crichton, P. Hanrahan, and K. Fatahalian, "Scanner: efficient video analysis at scale," *ACM Transactions on Graphics*, vol. 37, no. 4, pp. 1–13, Aug. 2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3197517.3201394

[51] "LosslessCut," 2023. [Online]. Available: https://github.com/mifi/lossless-cut

[52] "avcut," Feb. 2024, original-date: 2015-10-10T18:38:04Z. [Online]. Available: https://github.com/anyc/avcut

[53] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, G. Liu, A. Tao, J. Kautz, and B. Catanzaro, "Video-to-video synthesis," 2018.

[54] A. Mallya, T.-C. Wang, K. Sapra, and M.-Y. Liu, "World-Consistent Video-to-Video Synthesis," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, vol. 12353, pp. 359–378, series Title: Lecture Notes in Computer Science. [Online]. Available: https://link.springer.com/10.1007/978-3-030-58598-3_22

[55] W. Jo, G. Lim, G. Lee, H. Kim, B. Ko, and Y. Choi, "Vvs: Video-to-video retrieval with irrelevant frame suppression," 2023.

[56] B. Haynes, A. Mazumdar, A. Alaghi, M. Balazinska, L. Ceze, and A. Cheung, "Lightdb: A dbms for virtual reality video," *VLDB*, 2018.

[57] FFmpeg Developers, "Concatenating media files — FFmpeg wiki," 2023.

[58] A. Zabrovskiy, C. Feldmann, and C. Timmerer, "Multi-codec DASH dataset," *ACM Multimedia Systems*, 2018.