Optimizing Video Analytics with Declarative Model Relationships

Francisco Romero*
Stanford University
faromero@stanford.edu

Johann Hauswald*
Stanford University &
Sutter Hill Ventures
johannh@stanford.edu

Aditi Partap
Stanford University
aditi712@stanford.edu

Daniel Kang
Stanford University
ddkang@cs.stanford.edu

Matei Zaharia
Stanford University
matei@cs.stanford.edu

Christos Kozyrakis
Stanford University
christos@cs.stanford.edu

ABSTRACT

The availability of vast video collections and the accuracy of ML models has generated significant interest in video analytics systems. Since naively processing all frames using expensive models is impractical, researchers have proposed optimizations such as selectively using faster but less accurate models to replace or filter frames for expensive models. However, these optimizations are difficult to apply on queries with multiple predicates and models, as users must manually explore a large optimization space. Without significant systems expertise or time investment, an analyst may manually create an execution plan that is unnecessarily expensive and/or terribly inaccurate.

We propose *Relational Hints*, a declarative interface that allows users to suggest ML model relationships based on domain knowledge. Users can express two key relationships: when a model can replace another (CAN REPLACE) and when a model can be used to filter frames for another (CAN FILTER). We aim to design an interface to express *model relationships informed by domain specific knowledge* and define the constraints by which these relationships hold. We then present the *VIVA video analytics system* that uses relational hints to optimize SQL queries on video datasets. VIVA automatically selects and validates the hints applicable to the query, generates possible query plans using a formal set of transformations, and finds the best performance plan that meets a user's accuracy requirements. VIVA relieves users from rewriting and manually optimizing video queries as new models become available and execution environments evolve. We evaluate VIVA implemented on top of Spark and show that hints improve performance up to 16.6× without sacrificing accuracy.

PVLDB Reference Format:
Francisco Romero, Johann Hauswald, Aditi Partap, Daniel Kang, Matei Zaharia, and Christos Kozyrakis. Optimizing Video Analytics with Declarative Model Relationships. PVLDB, 16(3): 447 - 460, 2022.
doi:10.14778/3570690.3570695

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/stanford-mast/viva-vldb23-artifact.

*Denotes equal contribution.

Proceedings of the VLDB Endowment, Vol. 16, No. 3 ISSN 2150-8097.
doi:10.14778/3570690.3570695

1 INTRODUCTION

Video analytics, the ability to extract insights from video, is enabled by increasingly accurate machine learning (ML) models and access to large archives of professionally produced content or videos captured by devices like cellphones, security cameras, and video-conference systems. While we can already answer queries over videos like "have any cars passed this intersection that match an AMBER alert?", several challenges remain before video analytics are as practical and as performant over analytics on structured data. For complex video analytics queries with multiple predicates or ML models, users must manually optimize their queries to avoid the high cost of naively executing large models on every frame using expensive hardware. For example, it takes over 14 GPU-months to process 100 camera-months of video using a very accurate YOLOv5 model for object detection [55].

Consider an analyst studying political coverage of major cable news channels that writes a query to find instances of Bernie Sanders, a politician, reacting angrily to Jake Tapper, a TV news host [20]. Their query may use object detection to find scenes with two people, face recognition to find instances of Jake Tapper and Bernie Sanders, and emotion detection to detect angry reactions. This query can take minutes to execute using unnecessarily accurate models, even on small video inputs, making it challenging for the analyst to interactively explore their dataset. To improve performance, the analyst may use domain knowledge to explore the following model optimizations:

- *Replacement*: use a different model for a task, such as a cheaper but less accurate object detector [25, 27, 28, 47].
- *Input Filtering*: use a fast model to filter inputs to an expensive model [26, 36, 63]. For example, insert a binary classifier to detect faces before recognizing Tapper or Sanders in frames.
- *Predicate Reordering*: run emotion detection before face detection because it is more selective.

The domain knowledge needed to consider such optimizations may come from (1) historical or similar queries using alternate models, (2) insights about the training data or query dataset like knowing angry emotions are less prevalent than neutral or happy ones, or (3) knowledge about a general area of expertise (*e.g.,* news, traffic, or sports analysis) suggesting a particular fine-tuned model would be better suited for the domain.

Unfortunately, systems today do not provide an interface for users to specify optimizations based on domain knowledge. Users must manually explore the performance-accuracy tradeoff across numerous combinations of optimizations in queries with multiple predicates. We found that, for the news analysis query in which there are nearly 100 plan options, performance can vary by up to 11.7× across different query plans with an accuracy requirement of 80%. Systems today provide no easy way to validate potential optimizations either. Hence, in order to optimize video analysis queries, users must build significant expertise in ML models and systems, taking away valuable time and money from their primary task of gleaning insights from video data.

The first goal of this work is to design a user interface to express *model relationships informed by domain specific knowledge* and define the constraints by which these relationships hold. Our second goal is to develop a video query engine that *automatically validates relationships and optimizes complex queries*. The engine explores alternate query plans and handles performance-accuracy tradeoffs, relieving users from manual exploration and optimization.

We propose a declarative SQL interface for model relationships called *relational hints*. We capture the semantics of two relationships between a model M and model H considered for optimization:

- H CAN REPLACE M denotes H and M are interchangeable in the query plan. For example, H may be a faster object detection model the user wants to consider instead of the current one.
- H CAN FILTER M denotes H can be used to filter inputs to M. For example, H could be a fast binary classifier for detecting the presence of a face prior to recognizing a specific person.

Hints capture high-level relationships based on the models' output signatures and their class labels. Hence, they can often be reused across queries similar to an index. They remove the need for users to manually rewrite queries when a new model becomes available and reason about how the probabilistic nature of models impact their query's end-to-end accuracy goal.

We develop VIVA a video analytics system that optimizes complex SQL queries using relational hints. VIVA's hint validator first determines what hints are applicable for the query. VIVA's planner uses the validated hints to generate alternate plans using model replacements, data filtering, and predicate reordering while pruning and limiting the search space for fast query optimization. VIVA's optimizer enumerates plans enabled by hints and automatically navigates the performance-accuracy tradeoff to select the best performance plan meeting the user's accuracy requirements.

In summary, we make the following contributions:

- We highlight the difficulty of manually optimizing complex video queries, showing that performance on the same query applying different optimizations can vary by up to 11.7× (Section 2).
- We formalize a declarative SQL interface for users to specify intuitive relationships between ML models used in video analytics based on domain knowledge (Sections 3 and 4).
- We detail the design of VIVA that incorporates relational hints in query planning and optimization given the user's accuracy requirements (Section 5).
- We implement VIVA of Spark [64] (Section 6) and show that across four real-world queries with different video inputs, hints improve performance up to 16.6× while meeting user accuracy requirements (Section 7).

2 THE COMPLEXITY OF VIDEO QUERIES

Recent work on video analytics optimization focuses primarily on optimizing a single predicate that uses an ML model, implemented as a user-defined function (UDF) in a query execution engine. Current proposed techniques explore the performance-accuracy tradeoff using fast proxy models to replace more expensive ones [25, 63], cheap filters to reduce the amount of processing needed [26, 36] and indexing for video data [28]. In contrast, we focus on complex queries composed of multiple compute-intensive ML models and predicates. These queries are typically applied on large datasets using scale-out execution engines [42, 48]. Unfortunately, executing these complex queries with multiple ML models is prohibitively expensive and slow. Prior work estimates that a similar query to the TV News analysis query previously mentioned over one year of CNN videos would, at time of writing, take over 4 hours and cost more than $300 using cloud GPUs [31].

While it is possible to optimize in isolation each model and predicate in a complex query, this approach is unlikely to lead to best overall performance (lowest latency) and makes it difficult to achieve an overall accuracy goal. We use the TV News analysis query to illustrate the difficulty of manual optimization. We consider the impact of predicate reordering and two optimizations:

- **Model Replacement** – replace the original model with one that has the same input/output specification but a different performance accuracy profile. BlazeIt [25] and TASTI [28] are techniques that can generate very fast, purpose-built models but still require a user to manually specify the use of these fast models.
- **Data Filter Model** – run a cheap classifier ahead of a more expensive model to filter frames that are unlikely to satisfy the predicate. Systems like PP [36] and CORE [63] automate the insertion of filter models but do not take into account the effects of other optimizations from an end-to-end query perspective.

In the TV News analysis query, the analyst is looking for instances of the politician Bernie Sanders reacting angrily to a news anchor Jake Tapper. The analyst uses object detection to find frames with two people, face recognition to find frames with both Sanders and Tapper, and emotion detection to find the angry emotion. Figure 1 shows the accuracies and latencies of different plans for this query after applying the optimizations discussed above. Accuracy is calculated using F1 score [3, 5, 63] with respect to the original plan. The analyst sets an accuracy requirement of 80%.

Figure 1 shows that selecting the best of the 6 possible reorderings of the 3 predicates in the query leads to 5× latency improvement. Execution engines today treat UDFs as black-boxes that are not optimized by the execution engine [64]; the analyst must explore the impact of all possible orders. Next, we show the impact of manually considering model replacements. If the analyst uses a faster emotion detection (ED) model, latency improves by 1.01× but accuracy drops to 79%. In contrast, replacing object detection (OD) with a faster person detection model to label people in frames reduces latency by 1.2× over the best reordering without affecting accuracy (Best). Finally, we investigate the impact of using fast filter models, such as a cheap face detection (Haar) model to filter frames without faces [56] and a similarity detector (Sim) to remove frames that are not similar to a reference frame [4]. The Haar filter does not degrade accuracy but increases the latency by 1.5× over OD as its selectivity is low and acts as a poor filter. The Sim filter

Table 1: Model Relationship Matrix: dimensions to evaluate how to relate ML models. The result is the relationship in a query plan.

		Classes	
		Equal or Overlap	Disjoint
Signature	Equal	CAN REPLACE	CAN FILTER
	Not Equal	CAN FILTER	CAN FILTER

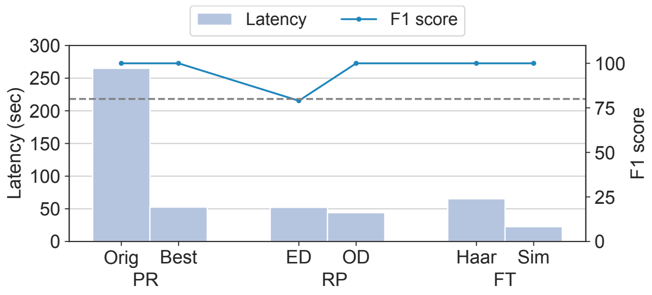


Figure 1: Manually Optimizing a TV News Analysis Query. PR: Predicate Reordering, RP: Replacing models, FT: Filters, dashed line denotes the user's accuracy requirement of 80%.

reduces latency by 2× over OD: it skips the expensive face recognition model for 94% of the frames without impacting accuracy. The key difference between the Haar and Sim filters is the former supports general face detection while the latter finds similarities to a reference frame.

The process of exploring the performance-accuracy tradeoffs and the interactions between these optimizations is long and cumbersome. Users must exhaustively study all options or use some ad-hoc trial and error process to find a good plan. For this query where the are 6 permutations, 2 replacements, and 2 filters, there are almost 100 plans to consider. The details to derive the number of plans using these optimizations are described Section 5.2. As new models and optimization opportunities arise, the number of plans for complex queries will only grow. Even for expert users, it is challenging to manually reason about this large number of query plans. It requires users to not only have an intuition about the potential of an optimization but also have deep knowledge of the performance-accuracy tradeoffs and selectivity of models within a large space of query plans.

3 DOMAIN SPECIFIC MODEL RELATIONSHIPS

There is potential for multiple models and predicates to improve the execution of a query. However, there is no framework to reason about the relationship between models. In this section, we propose a framework to define model relationships.

Suppose we have two models, M and H, that we use to process a set of frames F. The models emit a labeled frame with high confidence that satisfies a predicate or produce no output and the frame is dropped. The label is the output of the model assigned to the frame given its trained classes while the predicate is part of a user's query that filters by a specific label(s). For example, for an object detection model, the trained classes can be bus, car, person, *etc.* while the predicate can be to only return frames where the label is a person. This is a common scenario where a user runs a model that generates class labels and only wants to keep frames from a certain class. There are multiple execution plans that, with a given probability, can produce the same set of frames and labels on F:

- Plan A: Run M on F
- Plan B: Run H on F
- Plan C: Run H on F, then M on H's output
- Plan D: Run M on F, then H on M's output

We then ask the question: under what conditions do the above plans produce approximately the same results? For plans A and B to produce the same results, M and H must be interchangeable: H can replace M in the execution plan (or vice-versa). For Plans C and D to produce the same results as A and B, H can only drop frames M would also have dropped; H is a filter for M. We can characterize a model by its signature and output classes. The signature is the model's input and output specification. This is similar to terminology used by TensorFlow [54]. To compare two models, we ask the following questions:

- Are the model signatures equal or not?
- Do the models have equal, overlapping, or disjoint output classes?

Table 1 captures the different options along these dimensions. If H and M have equal signatures and equal or overlapping classes, then H can replace M. While two models can produce equivalent outputs, they may still differ in performance (execution latency) and/or accuracy. This type of model is referred to as a variant [46–48] or a proxy model [25]. The model architecture, dataset, and training parameters affect a model's performance and accuracy.

If H and M have equal signatures and disjoint classes, or their signatures are different, H can potentially filter frames for M. For example, consider an image classifier that outputs animal labels per image. Now consider an object detector that can produce the same class labels but the class is attributed to a bounding box. These two model signatures are not equal but there is overlap in the classes. The image classifier can be predicated on whether an animal was found. This only passes frames to the object detector that are highly likely to have an animal. The image classifier acts as a filter. The setup is similar for disjoint labels except a user specifies under what conditions the predicate for the image classifier is true. This can be specified based on a user's domain knowledge.

4 RELATIONAL HINTS

We next propose a declarative interface called *Relational Hints* (hints for short) that formalizes the model relationships defined in Table 1. Hints allow users to declaratively express domain specific knowledge about model relationships. The goal is to provide a query planner with information that enables alternate query plans. A planner can select among these plans to improve query performance or reduce the price while meeting a user's accuracy requirements. We first describe the different types of hints and their syntax. Next, we walk through a workflow of how hints are used in SQL queries. Finally, we describe sources informing the design of relational hints and relate them to well-known optimizations and domain intuition.

4.1 Relational Hint Types

A hint takes as input two models and a type, CAN REPLACE or CAN FILTER, that establishes a relationship between the models. We map Table 1 to our declarative hint interface.

Definition 1 *A relational hint is a user defined model relationship informed by domain knowledge for the purpose of suggesting alternate query plans to an optimizer.*

Similar to MicrosoftSQL hints [37] or MySQL hints [39], hints are options specified to the query optimizer to consider alternate query plans. Unlike the aforementioned hints, relational hints are not enforced. Users set a minimum query accuracy requirement and the optimizer chooses which hint(s) (if any) meet that requirement. The accuracy is in reference to the unmodified query plan where the labels produced by the original models represent the ground-truth. The accuracy is calculated on a video supplied by the user called a canary input. A canary input is a shorter clip that represents the type of events the user is looking for. A hint associates a hint model H to an original model M using model specific domain knowledge.

Definition 2 *Domain knowledge is external information about a model's signature and class labels in relation to another model.*

CAN REPLACE Hint. If a model H's signature and classes are equal or overlap with model M's signature and classes, a user can define a CAN REPLACE hint to suggest H can replace M in a plan:

```
CREATE HINT H CAN REPLACE M
  [ FALLBACK DISABLED | ENABLED ]
```

A CAN REPLACE hint is optionally parameterized by a `FALLBACK` argument. When disabled (the default), this expresses to the system that model H should completely replace M when processing frames. If enabled, the processing will fallback to the original model M if H does not produce a label because its confidence is too low. We assume confidence thresholds are pre-tuned and set for each model as is commonly done with existing optimizations [25]. In effect, this threshold arbitrates whether the model will generate a label or not. This can also be exposed to the user as a parameter to tune. Setting `FALLBACK ENABLED` may result in M having to process the same inputs that did not satisfy H's confidence threshold. This may negate some of the performance benefits of using H. However, it gives finer control to the user of the relationship and how much they want to trade-off performance and accuracy.

CAN FILTER Hint. If model H's signature is equal and its classes are disjoint from model M, or if model H's signature is not equal to model M's signature, a user can define a CAN FILTER hint to suggest that model H can filter frames for model M. Specifically, frames are only processed by M if they satisfy H's predicate with high confidence using the model's pre-set threshold:

```
CREATE HINT H CAN FILTER M
  [ CONDITIONED ON ANY | <list-of-classes> ]
```

A CAN FILTER hint is optionally parameterized by a `CONDITIONED ON` parameter which specifies the relationship between the model classes. By default, this parameter is set to `ANY`: any class in H can satisfy the condition. A user can optionally specify a list of classes. The list of classes means a user can condition M's input on the results of H's predicate as defined by the condition.

Prior work investigates automatically inferring relationships using historical data [36, 63]. Our interface could be extended to support automatic inference of these relationships by setting `CONDITIONED ON` to `AUTO`. In this work, we focus on the overall interface of expressing model relationships and leave it to future work to investigate inferring these relationships.

4.2 Example Workflow with Relational Hints

We now walk through a workflow using three relational hints for the TV News analysis query searching for Bernie Sanders reacting angrily to Jake Tapper Hints are registered once and automatically used on future queries when applicable. The first hint expresses knowledge that two object detection models have the same signature (labeled bounding boxes of objects) and generate the same number of classes but vary in performance and accuracy. These models can be related using a CAN REPLACE hint:

```
CREATE HINT ObjectDetectFast CAN REPLACE ObjectDetect
```

The second hint uses a tuned face recognition model trained on journalists and personalities. This is similar to a BlazeIt trained model to represent a more expensive model [18]. This model has the same signature (labeled bounding boxes of faces) as a general face recognition model, with some overlap in classes, including labels for Bernie Sanders and Jake Tapper. These models are again related using a CAN REPLACE:

```
CREATE HINT FaceRecogNews CAN REPLACE FaceRecognition
  FALLBACK ENABLED
```

The CAN REPLACE hint is parameterized with `FALLBACK ENABLED` to indicate the original FaceRecognition model should be used if FaceRecogNews does not emit a label due to low confidence.

The third hint considers a binary detector with labels face/no face. This binary detector can be trained using optimizations like Probabilistic Predicates [36]. Since frames with a face detected typically imply that a face is recognized, a user can express the following hint:

```
CREATE HINT FaceDetect CAN FILTER FaceRecognition
  CONDITIONED ON ['face']
```

Consider an analyst exploring a `VIDEO` table that contains frames from a TV dataset. They submit a query searching for instances of Bernie Sanders reacting angrily to Jake Tapper. The analyst sets an accuracy requirement of 90% and provides a short clip (the canary input) of Bernie Sanders being interviewed and reacting angrily to Jake Tapper. The system will use this video to estimate accuracy of new plans. In the following query, we **bold** models for which there are valid hints available:

```
SELECT frameID, EmotionDetect(frame) AS e,
       FaceRecognition(frame) AS f,
       COUNT(SELECT ObjectDetect(frame) AS o
         FROM VIDEO
         GROUP BY frameID
         WHERE o.label = 'person') AS pcount,
FROM VIDEO
WHERE e.label = 'angry' AND pcount = 2 \
  AND f.label LIKE '%Sanders%' AND f.label LIKE '%Tapper%'
ACCURACY 90%
```

The cost-based optimizer will generate additional plans using the registered hints and selects the fastest plan that meets the user accuracy requirement. The lowest cost plan that meets the user's

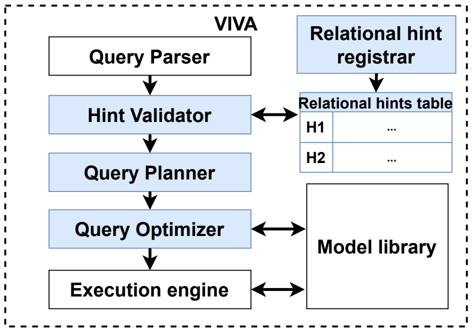


Figure 2: VIVA Architecture Diagram.

accuracy is the following, where the changes made are **bolded**:

```
Result
  + EmotionDetect.label isin ['angry']
    + ObjectDetectFast.pcount = 2 AND \
      ObjectDetectFast.label isin ['person']
      + FaceRecognition.label isin ['Tapper', 'Sanders']
          + FaceRecogNews.label isin ['Sanders'] \
            AND FaceRecogNews.conf > 0.8
              + FaceDetect.label isin ['face'] AND \
                FaceDetect.conf > 0.7
```

Note the optimizer in applying the hints also finds the optimial execution order to execute the most selective models first.

4.3 Sources of Relational Hints

We now give examples of where model relationships can originate from and how users can capture these relationships using hints.

Model Variants. Models having the same signatures and either equal or overlapping classes are ideal candidates for CAN REPLACE relationships. For example, when analyzing 5 popular open-source repositories for object detection models, we find there are at least 24 unique models with varying accuracy and performance characteristics [13, 45, 55, 57, 60]. Also, there are 16 pre-trained image classification models in PyTorch with varying performance and accuracy profiles [44]. These different profiles can come from the model architecture or the use of techniques like quantization that post-process the model to further trade accuracy for performance [23].

Additionally, an emerging trend is building models using a common set of layers and fine-tuning the rest for a specific application. The common layers are known as the "prefix" and the fine-tuned layers as the "suffix". These are also considered variants of the original model. A system can take advantage of saving the results of the prefix after the first call and reuse the results multiple times only needing to run the suffix model [24]. Model variants are best expressed as CAN REPLACE relationships.

Proxy Models. Proxy models can be trained to be smaller (in total GFLOPs) approximate versions of a larger, more accurate ML model. They can also be used to limit the number of invocations to the larger model. For example, a model may be trained on a subset of data because a fixed-view camera only needs to identify the original model's objects from a single viewpoint [25]. Other techniques like TASTI [28] train embedding indices at query optimization time. These indices run a model on a small fraction of the input dataset and store a representation (embeddings) of the frames and results from the model. At query time, the input frame embeddings are compared to the index and if the frames are similar enough, the stored results are used. This obviates the need to run an expensive model on the dataset. This techniques requires training an index for each target model.

Area Expertise. A practitioner could use their knowledge of the training data or the dataset to define even richer relationships. For example, a biologist may wish to detect bears or deer to study their foraging habits [9]. As an alternative way of detecting animals, the biologist knows camera trap feeds are mostly static and detecting motion is usually a good indication of an animal being present. A motion detection model has disjoint labels from an animal detection model. This can be expressed using a CAN FILTER hint:

```
CREATE HINT MotionDetect CAN FILTER AnimalDetect
  CONDITIONED ON ['motion']
```

Another example can be a sports analyst creating a highlight video of a basketball game. They can use an expensive action recognition model that does pose estimation to analyze if there was a scoring motion and if the ball went through the hoop. Alternatively, they could replace the action recognition model with an optical character recognition (OCR) model to detect score changes using a bounding box on the broadcast score [15, 40]. This would be cheaper because only a small section of the frame is analyzed. This can be expressed using a CAN REPLACE hint:

```
CREATE HINT ScoreChangeOCR CAN REPLACE ScoreActionRecog
```

Users can also express relationships where one of the models do not process frames. For example, consider the TV News analysis example from Section 1. An alternative way of detecting Bernie Sanders can be to search for him in video transcripts. This can be expressed as a CAN FILTER hint since it has disjoint labels from the face recognition model:

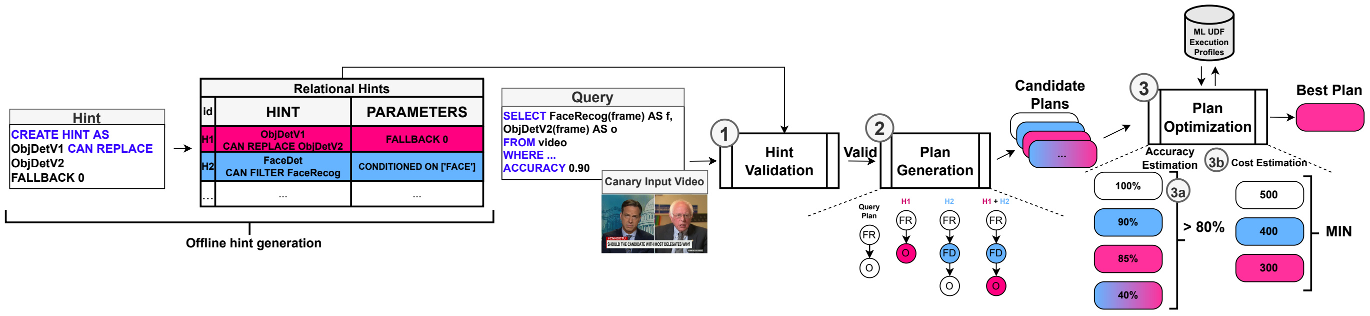
```
CREATE HINT TranscriptSearch CAN FILTER FaceRecognition
  CONDITIONED ON ['Sanders']
```

5 APPLYING RELATIONAL HINTS

We now describe the design of VIVA a video analytics system that interprets hints to optimize queries. VIVA enables users to query videos using SQL, provides an interface to specify hints and an accuracy threshold, and automatically applies hints to a query.

Figure 2 shows VIVA's system architecture. Blue components are specially designed for translating hints to executable plans. Users register hints with the registrar which are stored in the hints table. The parser processes the query and creates a query model tree. This tree is passed to the hint validator that determines which hints are relevant to the existing query. The planner and optimizer apply hint transformation rules to produce additional query plans, estimates accuracy, selectivity, and finally the overall cost of each plan. Depending on a user-defined optimization target (performance, cheapest price or best performance per dollar), VIVA selects the plan that meets the user's accuracy requirements. This plan is sent to the execution engine to process the user's input.

We walk through each blue component of Figure 2 by breaking down the query optimization steps shown in Figure 3. We design our system with the following goals in mind:


Hint

```
CREATE HINT AS
ObjDetV1 CAN REPLACE
ObjDetV2
FALLBACK 0
```

Relational Hints

id	HINT	PARAMETERS
H1	ObjDetV1 CAN REPLACE ObjDetV2	FALLBACK 0
H2	FaceDet CAN FILTER FaceRecog	CONDITIONED ON ['FACE']
...	...	...

Query

```
SELECT FaceRecog(frame) AS f,
ObjDetV2(frame) AS o
FROM video
WHERE ...
ACCURACY 0.90
```

Figure 3: VIVA Query Optimization Steps using Relational Hints.

Algorithm 1 Plan Generation with Hints

```
1:  $ValidHints \leftarrow ValidateHints(AllHints, Query)$
2:  procedure PLANNER($QueryTree$)
3:      for node in DepthFirstSearch(QueryTree) do
4:          $CandidateTrees.append($PERMUTATIONS$(node))$
5:          while $NumPlans != NewPlans$ do
6:              $CandidateTrees.append($APPLYHINTS$(node))$
7:              $NewPlans \leftarrow Len(CandidateTrees)$
8:              $NumPlans \leftarrow NewPlans$
9:          end while
10:     end for
11:     return $AllPlans$
12: end procedure
```

- **Validating hints (Section 5.1)** – The validator requires rules to arbitrate which hints can be used to generate additional plans.
- **Applying hints to generate plans (Section 5.2)** – The planner needs to translate validated hints to query plans using a formal set of transformation rules while limiting the number of total plans considered.
- **Providing accuracy guarantees (Section 5.4)** – Given a user's accuracy requirement, the optimizer must compare plans, and only select those that meet the accuracy requirement.

5.1 Validating Hints

After VIVA's parser produces a tree with the models to run, the hint validator determines which registered hints are valid for the query. This is Step ① in Figure 3. Throughout this section, we refer to the original model as M and the hint model as H. For both hint types, this is a fast static analysis where VIVA is comparing hint arguments to the model classes.

CAN REPLACE hints. CAN REPLACE is only applied if the overlap in classes of H and M are equal or larger than the classes predicated in the user's query. The requirement is the model H must produce the same predicated results as M. The hint is discarded if the classes are not equal or there is no overlap. To limit the search space, hint validation only retains hints that are likely to yield a plan with a lower cost than the original query ccomparing the profiled latency of the hint model to the original model. The latency is profiled offline as a one-time step.

CAN FILTER hints. VIVA validates that the user-specified classes in `CONDITIONED ON` are in model H. Because the classes can either overlap, be equal, or disjoint, there is no validation for M.

5.2 Generating Plans using Hints

We now describe the steps the planner takes to recursively apply hints to a query (Step ② in Figure 3).

Transforming a Query Plan. VIVA's parser transforms a user's query into an intermediate representation called a model tree. Each tree node represents a model and its predicate. If a parent node has a child, the output of the parent is dependent on its child (or children's) predicate(s) which preserves dependencies.

Algorithm 1 shows the planner's steps. The planner first reorders the predicates of that node's children (line 4) then applies the hints exhaustively (lines 5 ~ 8) until no new candidate trees can be generated at that node. The order the hints are applied does not matter. Remaining order-agnostic is necessary: a hint could generate a tree that could be further modified. This is Step ② of Figure 3, where we show a subset of the plans generated for the query after applying the hints from the table. A CAN REPLACE hint can modify a tree in two ways: (1) H replaces M in the new tree, (2) the planner will insert H before M if the user has set `FALLBACK ENABLED`. M will only run on frames where H did not produce a label because the confidence was too low. The predicate is applied to the union of H and M's results. For CAN FILTER hints, the planner will generate a tree where a model H predicated on `CONDITIONED ON` is inserted before model M. All inputs to M are predicated on H.

Enumerating Plans. To ensure all plans have been generated using hints, VIVA's planner analytically computes the number of expected plans based on the number of models and hints. The standard way of generating alternate query plans is to permute the predicates of independent models. Let N be the number of independent models at depth i. The number of valid plans is the product of the permutations of those models: $\prod_{j=1}^{i-1} N_j!$ For example, if a plan has three independent models A, B and C, there are six plans representing the permutations of these models (ABC, BAC, *etc.*). We also to consider cases where hints generate additional plans. A CAN REPLACE hint will only generate an alternate plan if either one or both models of the hint appear in the plan. A CAN FILTER hint will only generate an alternate plan if M appears in the plan where H will be inserted into the plan.

5.3 Canary Inputs

VIVA takes as input the query plan, an accuracy requirement and a canary input video. A canary input is a short video the user provides that represents events they are querying in the dataset.

Using small, representative canaries is common when tuning or calibrating computer vision, machine learning, and data mining systems before execution [35]. They have also been used by recent video analytics systems [59]. Canary data is similar to sampling during pre-processing to optimize the query [7] except in our case the user explicitly defines the data used. Empirically, we found it suffices to use a canary with at least one occurence of the event queried and some amount of noise to generate true positives and true negatives for calculating F1 score.

VIVA runs the original query plan on this input to generate ground-truth labels. Alternate query plans are also run on the canary and their accuracies are computed with respect to the ground-truth labels. The canary needs to closely resemble but not necessarily match the type of events a user is looking for. VIVA uses this to find which alternate plans will closely resemble the original plan *i.e.,* have high accuracy. The canary is specifically provided by the user and not sampled from the input dataset because it serves as a labelled set of frames, similar to a validation set.

In the TV News analysis example from Section 4.2, a canary input could contain scenes of Bernie Sanders upset in an interview with Jake Tapper or, more generally, scenes of two people being interviewed with one person reacting angrily. A poor canary has events for which the models in the original query plan would have low accuracy (*e.g.,* a basketball player dunking). In this work, canaries are manually selected to be representative of the query. In future work, we plan to explore techniques to automate the selection of canaries or notify users when their canary is not representative.

5.4 Selecting Plans to Meet User Requirements

The planner produces X plans $\mathcal{P} = \{P_1, P_2, \cdots P_X\}$. A plan P_x consists of N ordered models $\mathcal{M} = \{M_1, M_2, \cdots M_N\}$. The goal of VIVA's optimizer is to select the fastest or most price-efficient plan $\mathcal{P}^*$ that satisfies a user's accuracy requirement $\mathcal{A}$. We now discuss how VIVA's optimizer selects this plan.

Estimating Plan Accuracy. Relational hints enable users to express knowledge to the query optimizer to evaluate alternate query plans. Using hints, the planner may generate query plans that trade-off accuracy for performance. VIVA provides users the ability to express an accuracy target the optimizer must still meet in selecting the plan with the lowest cost. Step 3a of Figure 3 shows this step of the optimizer estimating accuracy for each plan.

We next detail our approach to accuracy estimation where we use common techniques from recent work for using the original models to generate ground-truth labels [25]. We use F1 score to estimate accuracy [3, 5, 63], and compute an F1 score per plan (not per model). To estimate each plan's accuracy, VIVA first runs the original models and candidate models over the canary input's frames and stores these results in a table. During query optimization, VIVA queries the table only with each plan's predicates to produce a final set of labels, R. This eliminates the need to run the model again for each plan. The results from the user's initial query plan are used as the ground-truth labels R_{truth}. Finally, the candidate plan's F1 score is computed using R_{truth} and R.

Selectivity and Cost Estimation. The last step in plan selection, Step 3b of Figure 3, is to estimate cost. VIVA determines how many frames f_i a model M_i needs to process. This is based on the selectivity s_{i-1} of the upstream model M_{i-1} and is given by: $f_i = M_{i-1} \times s_{i-1}$. We use a standard approach of estimating selectivity: VIVA samples a number of frames from the input dataset. We sample frames at a fixed rate from the input dataset similar to prior work; other techniques for sampling can also be used [3, 38]. VIVA estimates selectivity independently for each model.

VIVA's cost model is designed to support arbitrary backend hardware platforms and models with arbitrary batch sizes. The latency of executing M_i on hardware platform H_j for a batch of B inputs is $L_{H_j}^{M_i}(B)$. For each new model, VIVA profiles $L_{H_j}^{M_i}(B)$ once and stores its value for future cost estimations. If there is a data transfer time associated with a particular platform like the GPU, VIVA will profile transferring different batches of frames and builds a model to estimate transferring any number of frames. Profiling and building the model is a one-time, offline step. When estimating the cost, VIVA includes the data transfer time estimated by the model as part of the overall plan cost. For V different hardware platforms, there are up to N^V different hardware configurations that models in P_x can run on $\mathcal{K}_{P_x} = \{\mathcal{H}_{P_x}^1, \mathcal{H}_{P_x}^2, \ldots, \mathcal{H}_{P_x}^{N^V}\}$. A hardware configuration $\mathcal{H}_{P_x}^c = \{H_1, H_2, \ldots, H_N\}$ corresponds to a model M_i in plan P_x running on hardware platform H_i. Then, the estimated cost of running P_x with $\mathcal{H}_{P_x}^c$ is:

$$C(P_x, \mathcal{H}_{P_x}^c) = L_{Train} + \sum_{i=1}^{N} L_{H_i}^{M_i}(B) \times (f_i/B)$$

L_{Train} is the time to train models specific to the query. If models are trained in parallel, L_{Train} is the maximum time to train all models. If models are trained sequentially, L_{Train} is the sum of times to train all query-specific proxy models [25, 28]. If all models are available, $L_{Train} = 0$. The cost-optimal hardware configuration for models in P_x is:

$$\mathcal{H}_{P_x}^* = \arg \min_{\mathcal{H}_{P_x}^c \in \mathcal{K}_{P_x}} C(P_x, \mathcal{H}_{P_x}^c)$$

Finally, let A_x be the estimated accuracy for P_x. The cost-optimal plan, $\mathcal{P}^*$, among plans satisfying the user accuracy requirement is:

$$\mathcal{P}^* = \arg \min_{P_x \in \mathcal{P}} C(P_x, \mathcal{H}_{P_x}^*), \ s.t. \ A_x \geq \mathcal{A}$$

A user can parameterize the query with 3 targets: performance, cheapest price, or best performance per dollar. For performance, VIVA will return the fastest plan given the hardware available. For cheapest price, VIVA will return the cheapest plan based on estimated latency multiplied by the cost per hour. For best performance per dollar, VIVA will return the plan and target platform that delivers the highest end-to-end performance at the lowest price.

5.5 Plan Pruning

As the complexity of queries increases with more models, predicates, and hints, the number of possible plans generated grows exponentially. This increases VIVA's query optimization latency because of the larger search space to consider given plans generated using hints. We apply several heuristics and pruning techniques to limit the search space and provide fast query optimization.

At hint validation, VIVA finds any model that is more expensive than the original model in the user's query and removes it before the plan generation step. Using these models would generate plans that are strictly more expensive than the original query. During plan

Table 2: Queries, Datasets, Predicates, and Validated Hints Per Query.

Application	Dataset	Query Description	Predicates	# Hints
Traffic	Jackson square traffic camera [25, 28]	Cars turning left with people in intersection at night	time of day = night ∧ object = (people & car) ∧ object track	7
News	"Big three news" broadcasts [2, 20]	Jake Tapper interviewing angry Bernie Sanders	emotion = angry ∧ count(object = people) = 2 ∧ face = (Sanders & Tapper)	7
Sports	NBA games [21, 52]	LeBron James dunks	action = dunking basketball ∧ face = James	2
Bias	Casual conversations dataset [17]	Non-white females over the age of 19	age > 19 ∧ race != non-white ∧ gender = female	3

generation, VIVA eliminates redundant calls to models that could be a result of CAN REPLACE hints and push predicates down to the first call of the model. Thus, duplicate plans are eliminated which can occur as a result of hints replacing interchangeable models.

Our pruning approach is akin to recent work in video analytics optimization, CORE [63], which uses branch-and-bound to evaluate the plan cost after each model. CORE only retains plans that are likely to have a lower cost than the best plan found thus far. VIVA prunes a plan if 1) the accuracy requirement was already met with a lower accuracy model and the current plan uses a higher accuracy model, 2) the accuracy requirement was not met with a higher accuracy model and the current plan uses a lower accuracy model, or 3) if a plan's estimated cost exceeds the best complete plan's cost after a given model, it is pruned.

VIVA transparently applies these heuristics and pruning techniques which significantly limits the search space making query optimization fast. In future work, we plan to investigate other pruning techniques. For example, we can use statistics about CAN FILTER hints and their historical accuracy when generating plans. Historically low accuracy CAN FILTER hints can be pruned.

6 IMPLEMENTATION

VIVA is built on top of Spark [64] where users express queries using UDFs and predicates in SQL or Spark's dataframe API in Python. We use Spark's execution engine for UDF execution and take advantage of its optimizer for structured query optimizations. Video ingestion and indexing uses FFmpeg [11]. Frames are stored as raw byte arrays in a PySpark DataFrame to enable data pipelining.

We use pretrained PyTorch models for applications such as action recognition, object detection, and facial recognition [10, 41, 43, 45], and TensorFlow for bias analysis and emotion recognition [1, 53]. Computer vision models to detect day/night scenes, motion detection, and similarity detection are implemented using OpenCV [4]. For the day/night scene detector, we use Scikit-learn's Support Vector Machine (SVM) [50] implementation which we trained on 240 images of day/night frames from traffic camera feeds [25].

By giving users the ability to express relationships across a wide range of models, hints naturally capture several existing video analytics optimizations. We use cheaper, less accurate object detection models to represent techniques like BlazeIt [25] that propose training low accuracy purpose-built models. We implement three layer sharing models for race, gender, and age detection, using on DeepFace [51], that share a common set of layers like those produced by Mainstream [24]. CAN FILTER hints can be used to relate cheap binary classifiers such as those produced using techniques like Probabilistic Predicates [36] or CORE [63]. We use models of similar computational footprints to represent these optimizations. We use TASTI [28] to generate candidate models for CAN REPLACE or CAN FILTER hints. We use a pre-trained ResNet18 embedding model. Unless otherwise noted, these indexes are trained and available at query time.

7 EVALUATION

We now evaluate VIVA using the queries from Table 2. In all cases, the query plans benefit from the execution engine's query optimizer which applies standard structured query optimizations where possible. We deployed VIVA on Google Cloud Platform (GCP). We use a `n1-highmem-16` instance (16 vCPUs, 104 GB of DRAM). This instance features Intel Xeon E5-2699 v4 CPUs operating at 2.20GHz, Ubuntu 20.04 with 5.15.0 kernel. For GPU experiments, we consider `n1-highmem-16` instances with an NVIDIA T4 and a V100.

Queries and Datasets. Table 2 shows the queries, datasets, predicates, and total number of validated hints used to evaluated VIVA. All queries are complex: each with multiple models and predicates and represent a range of different applications. TV News analysis is based on queries used to explore a decade of US cable news [12, 20]. Sports analysis is commonly used for game planning, as well as for creating highlight reels [21]. Traffic analysis is used for landscaping and autonomous vehicle training [25, 26, 32]. Bias analysis is used to assess and detect model bias from training data [17].

We consider two inputs: the event searched is present in the video — `Event Present`, and no instances of the event are in the video — `Event not Present`. As shown in Table 2, these inputs come from the same dataset. All videos are one hour long, 360p, and are processed at 1 FPS. The framerate we use is chosen to be consistent with prior work [25]. The canary input is 15 seconds. We use F1 score for accuracy. Also consistent with prior work, selectivity estimation is performed over 3% of the input frames [6].

Relational Hints. Table 3 shows example tasks, models, and hints we use in our evaluation. In total, we use 19 different hints — 11 CAN REPLACE, 4 CAN REPLACE with `FALLBACK ENABLED`, and 4 CAN FILTER— across 30 different models. We capture several sources of domain knowledge in our choice of hints.

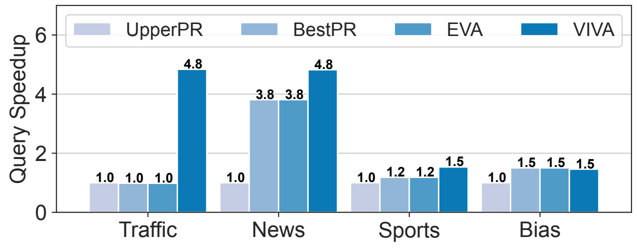
Model Variants. Using a smaller object detection as a replacement for a larger model is a classic example of a model variant where the models have the same output and classes but have different model architectures *e.g.,* SmallObjDet CAN REPLACE LargeObjDet. For layer sharing models, a user specifies the suffix layers to run in the model relationship *e.g.,* RaceID CAN REPLACE SuffixRaceID. VIVA automatically determines whether it is worthwhile to execute the combination of prefix and suffix layers.

Proxy Models. Cheaper, less accurate models generated using TASTI can be used as a replacement for more expensive ones like face detection and action detection. To indicate that these models should use the larger models when a label cannot be produced, we set `FALLBACK ENABLED` for CAN REPLACE hints.

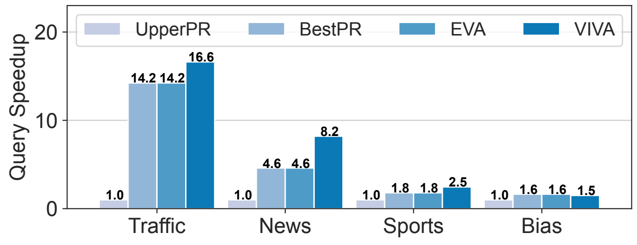
Area Expertise. Classical computer vision techniques can be used by practitioners and expressed as model relationships. For example,

Table 3: Tasks, Models, and Sample Relational Hints.

Task	Models	Relational Hints
Emotion Detection	MTCNN Emotion Detection, HAAR Emotion Detection, TASTI Emotion Detection	MTCNNEmoDet **CAN REPLACE** HAAREmoDet, TASTIEmoDet **CAN REPLACE** MTCNNEmoDet FALLBACK ENABLED
Object Detection	Small Object Detection, Large Object Detection, Object Similarity Detection, Motion Detection	SmallObjDet **CAN REPLACE** LargeObjDet, ObjSimDet **CAN FILTER** LargeObjDet, MotDet **CAN FILTER** LargeObjDet `CONDITIONED ON ['motion']`
Image Classification	ResNet50 Image Classification, ResNet18 Quantized Image Classification	QImgCls **CAN FILTER** LargeObjDet
Facial Recognition	Face Recognition, TASTI Face Recognition	TASTIFaceRecog **CAN REPLACE** FaceRecog FALLBACK ENABLED
Race Identification	RaceID, Suffix RaceID	RaceID **CAN REPLACE** SuffixRaceID
Action Recognition	Action Recog, Action Similarity Recog	ActionSimDet **CAN FILTER** ActionRecog
Day/Night Detection	Pixel Brightness Detect, SVM for Day/Night Detection	PixelBriDet **CAN REPLACE** SVM



(a) `Event Present` in frames.



(b) `Event not Present` in frames.

Figure 4: Query Speedup Relative to `UpperPR`.

a pixel brightness detector can be used as a replacement for an SVM model trained to detect whether a frame is from day or night (PixelBriDet CAN REPLACE SVM). Performing similarity detection to find frames similar to a reference image can be used as a replacement for detecting actions. In the case of the Traffic analysis query, since the analyzer is querying static camera feeds, they can infer that detecting cars and people can be cheaper using motion detect. They can relate motion detection to object detection as: MotDet CAN FILTER LargeObjDet `CONDITIONED ON ['motion']`.

We compare VIVA to the following baselines:

- `Upper Bound Predicate Reorder (UpperPR)`: this is the worst-case latency of predicate reordering for a given accuracy requirement if a system does not support selectivity and cost estimation for ML UDFs which is common in today's execution engines.
- `Best Predicate Reorder (BestPR)`: this represents what a user can expect if a video analytics system is able to do selectivity and cost estimation for ML UDFs to find the lowest latency ordering given an accuracy requirement.
- `EVA`: a recently-proposed, state-of-the-art video analytics system whose optimizer makes model and predicate reordering selections given a fixed accuracy [62]. Users specify a model's accuracy using coarse-grained indicators: low for accuracies 80% and below, medium for accuracies [80%, 90%), and high for accuracies 90% and above. During query optimization, `EVA` selects each model to use separately based on the plan accuracy requirement.

7.1 Improving Query Performance

We first explore how VIVA uses hints to improve performance over predicate reordering with an accuracy requirement of 90%. Figure 4 shows the performance for each query for `Event Present` (Figure 4a) and `Event not Present` (Figure 4b). The latencies presented are inclusive of query optimization time. Table 4 shows the plan `UpperPR` uses and the best plan VIVA uses and its accuracy.

Traffic Analysis. For `Event Present`, `UpperPR` filters by time of day, objects, and object tracking. `EVA` and `BestPR` filter by time of day last. Since `Event Present` is all night scenes, no filtering occurs with the SVM day/night detection. VIVA uses a pixel brightness detection and a faster object detection model than `EVA` since its accuracy estimator determines it can use what `EVA` considers a "low" accuracy model. This enables VIVA to improve performance by 4.8× over the baselines. For `Event not Present`, VIVA and `EVA` first filter by time of day since this input is all day scenes. VIVA is slightly faster (1.2×) because it uses the pixel brightness detection. `UpperPR` runs the time of day detection last, which leads to a 16.6× drop in performance compared to VIVA.

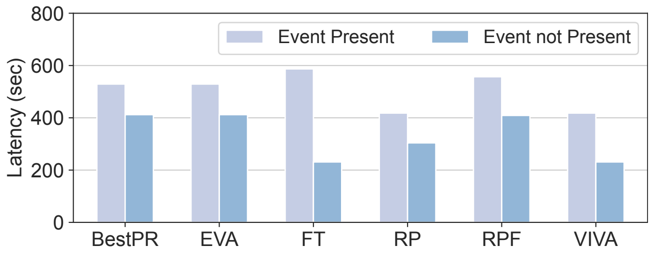
News Analysis. For `Event Present`, `UpperPR` first filters by emotion, which is the least efficient since this expensive model must process all frames. `EVA` and `BestPR` first filter by faces — a faster model — before doing object and emotion detection, respectively. VIVA uses a faster object detection (which `EVA` would classify as low accuracy), along with a TASTI-trained model for emotion detection. The TASTI-trained emotion detection is backed by the HAAR emotion detection. This improves performance 4.8× over `UpperPR`, and 1.3× over `EVA` and `BestPR`. For `Event not Present`, VIVA uses object similarity detection as a result of a CAN FILTER hint.

Sports Analysis. For `Event Present`, all baselines use the same two models, with `EVA` and `BestPR` benefiting from predicate reordering. VIVA uses a TASTI-trained action detection backed by the original action detection model. This enables VIVA to improve performance by 1.5× over `UpperPR`, and 1.2× over `EVA` and `BestPR`. For `Event not Present`, VIVA uses a similarity detection for detecting dunks from a reference image. This improves performance up to 2.5×.

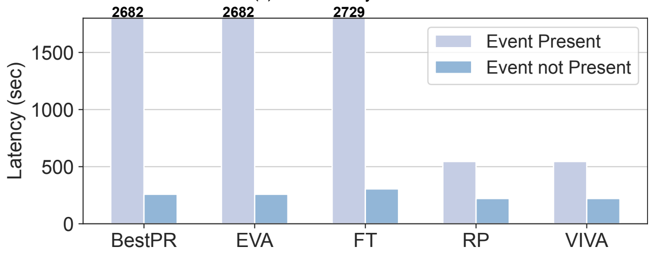
Bias Analysis. For `Event Present`, VIVA uses a plan with common prefix layer models for race and age detection, specified using a CAN REPLACE hint. The common layers are run once and reused for the two suffix models. This improves performance by 1.5× over `UpperPR`, and matches the performance of `EVA` and `BestPR`. For `Event not Present`, VIVA does not use the common prefix layer models, since the gender detection model can filter the majority of

Table 4: Best Plan Identified by VIVA. PR: Predicate Reorder, RP: CAN REPLACE, RPF: CAN REPLACE with `FALLBACK ENABLED`, FT: CAN FILTER.

Application	Original Plan	Best Hint Plan: ∃: `Event Present`, ∄: `Event not Present`	Accuracy
Traffic	TimeOfDay ∧ Object ∧ ObjectTrack	∃: RP(Object) ∧ ObjectTrack ∧ RP(TimeOfDay)	100%
		∄: RP(TimeOfDay) ∧ RP(Object) ∧ ObjectTrack	100%
News	Emotion ∧ Object ∧ Face	∃: RP(Object) ∧ Face ∧ RPF(Emotion) ∧ RP(Emotion)	91%
		∄: FT(Object) ∧ Object ∧ Face ∧ Emotion	91%
Sports	Action ∧ Face	∃: Face ∧ RPF(Action) ∧ Action	100%
		∄: FT(Action) ∧ Action ∧ Face	90%
Bias	Age ∧ Gender ∧ Race	∃: Gender ∧ RP(Race) ∧ RP(Age)	100%
		∄: Gender ∧ Age ∧ Race	100%



(a) News analysis.



(b) Traffic analysis.

Figure 5: Query Latency when Ablating Hints. RP: CAN REPLACE, RPF: CAN REPLACE with `FALLBACK ENABLED`, FT: CAN FILTER.

Table 5: Query Optimization Latencies for Figure 4a Queries.

Application	# Plans w/o Pruning	# Pruned Plans	Query Opt. (% Total)	Query Exec. (% Total)	Total
Traffic	60	17	92s (17%)	453s (83%)	545s
News	432	25	116s (28%)	302s (72%)	418s
Sports	6	6	130s (18%)	592s (82%)	722s
Bias	42	24	88s (16%)	473s (84%)	561s
		Average	107s (20%)	455s (80%)	562s

frames. VIVA is slightly slower in this case (1.1×) compared to `EVA` and `BestPR` since both run the same plan but VIVA additionally performs accuracy estimation. Performance is similar across the board for this query because the original models and the CAN REPLACE suffix models have similar performance.

7.2 Query Optimization Latency

We next evaluate VIVA's query optimization latency. During query optimization, VIVA estimates hint-generated plan accuracies and selectivities. This time increases with the number of independent query predicates and the number of applicable hints per query. Table 5 shows the absolute and relative time breakdowns for query optimization and execution using `Event Present` (a one hour input) with the 15 second canary input.

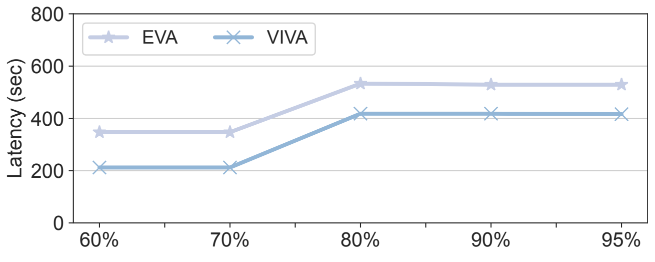
VIVA spends on average 20% of execution time on query optimization. This is in line with recently released systems like FiGO [6], MIRIS [3], and Jellybean [58] where 20%-25% of query execution is spent on optimization. Pruning eliminates on average 70% of plans for 3 out of 4 queries. The Sports application does not benefit from pruning because of the small number of hints used and relatively small number of additional plans generated. Without pruning, the News analysis query's optimization time is 2.1× higher. Query execution time represents the majority of the time for all queries: 80% on average, up to 84%. For larger inputs assuming the same query, the time spent on query execution will grow while query optimization stays constant. Lastly, query optimization time varies only by up to 50% across all queries despite the number of plans differing by up to 72×. This shows VIVA can scale as queries become more complex and the number of hints grows.

7.3 Performance Impact of Hint Types

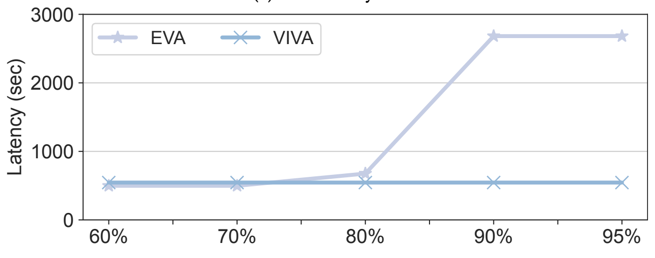
We next analyze what hints improved performance by ablating the registered hints VIVA applies. We use the News and Traffic queries. For each query, we consider each available hint type separately (RP: CAN REPLACE, RPF: CAN REPLACE with `FALLBACK ENABLED`, FT: CAN FILTER) and VIVA using all available hints). Traffic analysis has no CAN REPLACE with `FALLBACK ENABLED`.

News Analysis. Figure 5a shows the ablated performance for `Event Present` and `Event not Present`. In both cases, the best performance comes from using a mix of hints. For `Event Present`, `RP` uses a faster object detect. `RPF` uses TASTI-trained models for emotion and object detect but uses the more expensive object detection as a fallback model. Interestingly, VIVA selects a different predicate ordering in each case: object detection runs first for `RP` while face detection runs first with `RPF`. `RP` is faster than `EVA` and `BestPR` since `RP` identifies a faster but lower accuracy object detection that can meet the accuracy requirement compared to the high accuracy one used by `EVA`. `FT` uses the same plan as `EVA` and `BestPR` since using CAN FILTER hints do not meet the accuracy requirement. For `Event not Present`, VIVA uses an object similarity detection model to improve performance.

Traffic Analysis. Figure 5b shows the ablated performance for `Event Present` and `Event not Present`. For both inputs, the best plan that meets the accuracy requirement is to use only CAN REPLACE hints. For `Event Present`, `FT` again picks the same plan as `EVA` since the motion detect model specified using the CAN FILTER hint does not meet the accuracy requirement. Plans are closer in performance for `Event not Present` since filtering by



(a) News analysis.



(b) Traffic analysis.

Figure 6: Query Latency as Accuracy Requirements Vary.

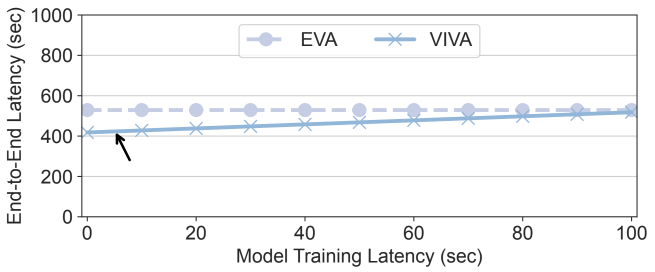


Figure 7: Query Latency of News Analysis with Training Latency. Arrow: TASTI index creation.

time of day first is best in all cases. RP and VIVA use pixel brightness detection to improve performance over EVA and FT.

7.4 Trading Off Latency and Accuracy

We now evaluate how VIVA automatically chooses the highest performance plan as the accuracy requirement varies. Using the News and Traffic analysis queries, we sweep accuracy requirements from 60% to 95% and use the `Event Present` input. EVA uses low accuracy models for requirements 80% and below, medium accuracy models for [80%, 90%) requirements, and high accuracy models for requirements 90% and above.

News Analysis. Figure 6a shows the results of varying accuracy where, aexpected, more stringent accuracy requirements also result in a decrease in performance. For accuracy requirements of 80% and 90%, VIVA selects the plan shown in Table 4. For accuracy requirement of 95%, VIVA uses the faster object detection model, but no longer uses TASTI-trained models or the faster emotion detect. However, the performance is similar to the plan shown in Table 4. The performance difference between these plans is 1.8×. VIVA outperforms EVA for all accuracy requirements (up to 1.5×) since it identifies the best performing combination of hints that meet the accuracy requirements. Indeed, for accuracy requirements of 90% and 95%, VIVA uses faster models that meet the accuracy requirements, while EVA uses the slower, high accuracy models.

Traffic Analysis. Figure 6b shows the results where VIVA identifies the plan shown in Table 4 meets all accuracy requirements, and hence uses the same plan in all cases. EVA has similar performance for low accuracies, but uses increasingly larger object detection models as the accuracy requirement becomes more stringent. This enables VIVA to improve performance over EVA by up to 4.8×.

It can be difficult to know when models should be updated or re-trained using optimizations like TASTI and BlazeIt. By selecting to use these models for lower accuracy requirements, but not for more stringent ones, VIVA can guide training decisions.

7.5 Impact of Training and Indexing

We next investigate how plan selection can be impacted by the need to construct an index or train a model for replacement or filtering at query time. This results in additional cost from training or indexing to plans that include hints with CAN FILTER or CAN REPLACE with `FALLBACK ENABLED`. We use the setup from Section 7.1 and focus on the News analysis query. We vary the training latency from 0sec (already exists) to 100sec in increments of 10sec.

Figure 7 shows the end-to-end latency (y-axis) for the two baselines and VIVA as the training latency varies (x-axis). We note VIVA matches or outperforms EVA even when spending up to a minute for training, since it spends less time on query execution. The arrow shows the case for creating a TASTI [28] index, which is on the order of seconds if frame embeddings are available. Proxy models can be trained in tens to hundreds of seconds [25], which can still be worth this additional upfront cost. Furthermore, caching this model means VIVA only incurs a one-time training cost that can benefit future queries as well. As noted in Section 5.4, VIVA considers this training time when selecting the best plan to execute.

7.6 Optimizing Across Hardware Platforms

We now evaluate VIVA's ability to generate and compare plans for different hardware platforms. We consider three instances: a standalone `n1-highmem-16` (CPU), a `n1-highmem-16` with a T4 GPU, and a `n1-highmem-16` with a V100 GPU. We use the GCP pricing for each: 0.66 $/hr for CPU, 0.91 $/hr for T4, and 2.40 $/hr for V100 [14]. We use out-of-the-box GPU implementation and fallback to CPU implementations if not available on the GPU. We study three optimization goals: performance (fastest plan), cheapest price, or best end-to-end performance per dollar. We use the Traffic and TV News queries on the `Event Present` input.

Figure 8 shows the results where VIVA optimizes for performance with the final dollar cost of the plan annotated. For Traffic analysis, the T4 GPU is 1.8× faster than the CPU while being 30% cheaper. While the T4 GPU instance is more expensive, the faster execution means the instance can be provisioned for less time. Similarly for News analysis, execution with the T4 is ~2× faster and 42% cheaper. In both cases, the V100's performance improvement of ~2× does not outweigh its high cost, 1.8× more expensive compared to a CPU. As shown in Table 6, the optimizer chooses the same plan in all cases since object detection can be significantly accelerated on GPU compared to running on CPU and the latency is the only variable when estimating cost. In this study, we do not

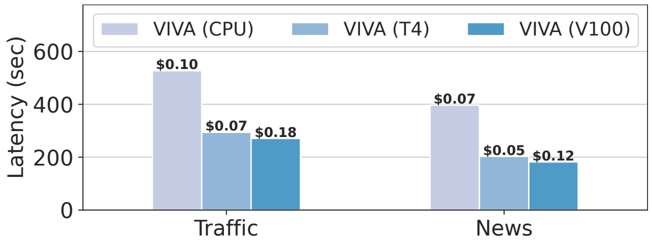


Figure 8: VIVA Execution using the CPU, T4 GPU, and V100 GPU. Latencies presented are the fastest plans given the available hardware.

Table 6: Hardware Platform Selection. Perf./$ normalized to CPU. Chosen HW is bolded. ***Model executes on CPU if GPU selected.**

App.	Opt. Target (HW avail.)	Selected Plan	Perf./$
Traffic	Perf. (CPU, T4, **V100**)	RP(Obj.) ∧ ObjTrack ∧ RP(TimeOfDay)*	1.04
	Cost (CPU, **T4**)		2.33
	Cost (**CPU**, V100)		1.00
News	Perf. (CPU, T4, **V100**)	RP(Obj.) ∧ Face ∧ RPF(Emo.)* ∧ RP(Emo.)	1.31
	Cost (CPU, **T4**)		2.23
	Cost (**CPU**, V100)		1.00

include mixed precision models that could take advantage of half-precision units on the GPU. Such models can be defined as CAN REPLACE hints and be considered during query optimization.

Table 6 shows the results when VIVA optimizes for cost. VIVA selects a different hardware platform depending on hardware availability. Consistent with our previous results, when optimizing for dollar cost, VIVA will favor a CPU plan if the accelerator available is the V100 and favor the T4 over the CPU. When optimizing for performance per dollar, VIVA will choose the T4 plan since it is up to 2.3× better than the plans for CPU and V100.

8 RELATED WORK

Accelerating Queries via Specialization. A large body of work uses cheap approximations to accelerate specific classes of queries, ranging from selection [26, 27, 36, 63], aggregation [25], and aggregation with predicates [29]. There is also work on using embedding indexes as cheap approximations [19, 28]. VIVA is the first system to provide a general interface, hints, that captures these optimizations and their impact on complex query performance and accuracy.

Video Frame Sampling. Several projects have focused on decreasing the amount of data models need to process via dynamic sampling rates. MIRIS [3] executes object detection and object tracking at reduced framerates and increases the framerate for low confidence detections. ExSample [38] splits a video dataset into temporal chunks and prioritizes processing chunks with higher probabilities of finding a new object. It iteratively updates its estimates as more frames are processed by leveraging an adaptive sampling algorithm based on Thompson sampling [49]. Depending on the query type, varying the sampling rate can affect the accuracy since lower sampling rates may lead to missed objects. The optimizations enabled by hints are orthogonal to existing sampling techniques.

Optimizing ML Execution and Storage. Systems such as Scanner [42], VideoStorm [65], and Llama [48] have focused on optimizing DNN execution by efficiently utilizing hardware resources for execution plans for video analytics. The scale-out and serverless techniques underpinning these systems are complementary to optimization with hints. Hence these systems can be integrated into, or used instead of VIVA's execution engine to further accelerate queries. Several recent projects have also focused on optimizing aspects of video retrieval from storage and how video data are stored and decoded [8, 16, 30, 61]. These techniques are also important for end-to-end efficiency but are complimentary to this paper's focus.

Functional Dependencies. Functional dependencies [22, 34] help database designers automatically determine the relation of one attribute to another. However, existing work is limited to structured data that can be easily analyzed to determine relationships. Video analytics queries execute expensive DNNs over unstructured records, which makes it infeasible to infer the relationships without first materializing the results. Hints enable VIVA to consider additional query plans that can improve performance and cost without having to first materialize the results.

Specifying Domain Knowledge. Providing extra knowledge to a system to improve query execution is an idea with roots in the early days of query processing. Hints most closely resemble early work in semantic integrity constraints [33], and more generally hints in existing database systems, such as MicrosoftSQL hints [37] and MySQL hints [39]. A key difference from domain knowledge for structured data is that ML models are probabilistic and require a system to consider and provide accuracy guarantees. VIVA reasons about the accuracy impact on plans using hints.

9 CONCLUSION

In this paper, we addressed the challenge of users having to manually explore performance-accuracy tradeoffs across combinations of optimizations in video analytics queries with multiple predicates. We proposed relational hints, a declarative interface to express ML model relationships, informed by domain specific knowledge. Relational hints eliminate the need for users to manually rewrite their queries when a new model becomes available and manually reason about how the use and order of the various models available impact their query's performance and accuracy. To determine how and when relational hints can be used to optimize queries, we designed the *VIVA video analytics system.* VIVA uses hints that are validated for each query to generate additional query plans using a formal set of transformations, and selects the best performance plan that meets user accuracy requirements. Using relational hints, we show that VIVA over Spark improves performance up to 16.6× without sacrificing accuracy for a range of complex queries.

ACKNOWLEDGEMENTS

We thank the Stanford Platform Lab and its affiliates (Cisco, Facebook, Google, Nasdaq, NEC, VMware, and Wells Fargo), the Open Philanthropy project, and Sutter Hill Ventures. We also thank affiliates of the Stanford DAWN project—Ant Financial, Facebook, Google, and VMware—as well as Toyota Research Institute ("TRI"), 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 NSF. TRI provided funds to assist the authors with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity. Francisco Romero was supported by a Stanford DARE Fellowship.

REFERENCES

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *OSDI*.

[2] Internet Archive. 2022. TV News Archive. https://archive.org/details/tv.

[3] 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*.

[4] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[5] Jiashen Cao, Ramyad Hadidi, Joy Arulraj, and Hyesoon Kim. 2021. THIA: Accelerating Video Analytics using Early Inference and Fine-Grained Query Planning. arXiv:2102.08481

[6] Jiashen Cao, Karan Sarkar, Ramyad Hadidi, Joy Arulraj, and Hyesoon Kim. 2022. FiGO: Fine-Grained Query Optimization in Video Analytics. In *SIGMOD*.

[7] Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. 2017. Approximate query processing: No silver bullet. In *SIGMOD*.

[8] Maureen Daum, Haynes Brandon, Dong He, Amrita Mazumdar, and Magdalena Balazinska. 2021. TASM: A Tile-Based Storage Manager for Video Analytics. In *ICDE*.

[9] Maureen Daum, Enhao Zhang, Dong He, Magdalena Balazinska, Brandon Haynes, Ranjay Krishna, Apryle Craig, and Aaron Wirsing. 2022. VOCAL: Video Organization and Interactive Compositional AnaLytics. In *CIDR*.

[10] Tim Esler. 2022. InceptionResNet Face Recognition in PyTorch. https://github.com/timesler/facenet-pytorch

[11] FFmpeg. 2022. FFmpeg. https://ffmpeg.org/.

[12] 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:1910.02993

[13] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. 2021. YOLOX: Exceeding YOLO Series in 2021. arXiv:2107.08430

[14] Google. 2022. Google Compute Engine: GPUs Pricing. https://cloud.google.com/compute/gpus-pricing.

[15] Google. 2022. Tesseract OCR. https://github.com/tesseract-ocr/tesseract.

[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*.

[17] Caner Hazirbas, Joanna Bitton, Brian Dolhansky, Jacqueline Pan, Albert Gordo, and Cristian Canton Ferrer. 2021. Towards Measuring Fairness in AI: the Casual Conversations Dataset. arXiv:2104.02821

[18] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *ICCV*. IEEE.

[19] Wenjia He, Michael R. Anderson, Maxwell Strome, and Michael Cafarella. 2020. A Method for Optimizing Opaque Filter Queries. In *SIGMOD*.

[20] James Hong, Will Crichton, Haotian Zhang, Daniel Y. Fu, Jacob Ritchie, Jeremy Barenholtz, Ben Hannel, Xinwei Yao, Michaela Murray, Geraldine Moriba, Maneesh Agrawala, and Kayvon Fatahalian. 2021. Analysis of Faces in a Decade of US Cable TV News. In *SIGKDD*.

[21] James Hong, Matthew Fisher, Michaël Gharbi, and Kayvon Fatahalian. 2021. Video Pose Distillation for Few-Shot, Fine-Grained Sports Action Recognition. In *ICCV*.

[22] Ihab F. Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. 2004. CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies. In *SIGMOD*.

[23] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*.

[24] Angela H Jiang, Daniel L-K Wong, Christopher Canel, Lilia Tang, Ishan Misra, Michael Kaminsky, Michael A Kozuch, Padmanabhan Pillai, David G Andersen, and Gregory R Ganger. 2018. Mainstream: Dynamic {Stem-Sharing} for {Multi-Tenant} Video Processing. In *ATC*.

[25] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics. In *PVLDB*.

[26] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: optimizing neural network queries over video at scale. In *PVLDB*.

[27] Daniel Kang, Edward Gan, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2020. Approximate Selection with Guarantees using Proxies. In *PVLDB*.

[28] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2022. Semantic Indexes for Machine Learning-based Queries over Unstructured Data. In *SIGMOD*.

[29] Daniel Kang, John Guibas, Peter Bailis, Yi Sun, Tatsunori Hashimoto, and Matei Zaharia. 2021. Accelerating Approximate Aggregation Queries with Expensive Predicates. In *PVLDB*.

[30] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia. 2021. Jointly optimizing preprocessing and inference for DNN-based visual analytics. In *PVLDB*.

[31] Daniel Kang, Francisco Romero, Peter Bailis, Christos Kozyrakis, and Matei Zaharia. 2022. VIVA: An End-to-End System for Interactive Video Analytics. In *CIDR*.

[32] Fiodar Kazhamiaka, Matei Zaharia, and Peter Bailis. 2021. Challenges and Opportunities for Autonomous Vehicle Query Systems. In *CIDR*.

[33] Jonathan J King. 1979. *Exploring the Use of Domain Knowledge for Query Processing Efficiency.* Technical Report. Stanford University, Dept of Computer Science.

[34] Sebastian Kruse and Felix Naumann. 2018. Efficient Discovery of Approximate Dependencies. In *PVLDB*.

[35] Michael A Laurenzano, Parker Hill, Mehrzad Samadi, Scott Mahlke, Jason Mars, and Lingjia Tang. 2016. Input responsiveness: using canary inputs to dynamically steer approximation. In *PLDI*.

[36] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. 2018. Accelerating Machine Learning Inference with Probabilistic Predicates. In *SIGMOD*.

[37] Microsoft. 2022. MicrosoftSQL Hints (Transact-SQL). https://docs.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql?view=sql-server-ver15.

[38] Oscar Moll, Favyen Bastani, Sam Madden, Mike Stonebraker, Vijay Gadepally, and Tim Kraska. 2022. ExSample: Efficient Searches on Video Repositories through Adaptive Sampling. In *ICDE*.

[39] MySQL. 2022. MySQL Optimizer Hints. https://dev.mysql.com/doc/refman/8.0/en/optimizer-hints.html.

[40] Scoreboard OCR. 2022. Scoreboard OCR. https://scoreboard-ocr.com/start.

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*.

[42] Alex Poms, William Crichton, Pat Hanrahan, and Kayvon Fatahalian. 2018. Scanner: Efficient Video Analysis at Scale. In *SIGGRAPH*.

[43] PyTorch. 2022. 3D ResNet Video Classification in PyTorch. https://pytorch.org/hub/facebookresearch_pytorchvideo_resnet/

[44] PyTorch. 2022. Image Classification: Models and Pre-Trained Weights. https://pytorch.org/vision/stable/models.html#classification

[45] PyTorch. 2022. Object Detection: Models and Pre-Trained Weights. https://pytorch.org/vision/stable/models.html#object-detection-instance-segmentation-and-person-keypoint-detection

[46] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. 2020. MLPerf Inference Benchmark. In *ISCA*.

[47] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated Model-less Inference Serving. In *ATC*.

[48] Francisco Romero, Mark Zhao, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. Llama: A Heterogeneous & Serverless Framework for Auto-Tuning Video Analytics Pipelines. In *SoCC*.

[49] Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. 2017. A Tutorial on Thompson Sampling. arXiv:1707.02038

[50] Scikit. 2022. Support Vector Classifier in Scikit. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[51] Sefik Ilkin Serengil and Alper Ozpinar. 2021. HyperExtended LightFace: A Facial Attribute Analysis Framework. In *ICEET*.

[52] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. 2019. Nexus: A GPU Cluster Engine for Accelerating DNN-Based Video Analysis. In *SOSP*.

[53] Justin Shenk. 2022. Facial Expression Recognition. https://github.com/justinshenk/fer

[54] TensorFlow. 2022. Signatures in TensorFlow Lite. https://www.tensorflow.org/lite/guide/signatures.

[55] Ultralytics. 2022. Yolov5 Object Detection in PyTorch. https://github.com/ultralytics/yolov5.

[56] Paul Viola and Michael Jones. 2001. Rapid object detection using a boosted cascade of simple features. In *CVPR*.

[57] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. 2021. You Only Learn One Representation: Unified Network for Multiple Tasks. arXiv:2105.04206

[58] Yongji Wu, Matthew Lentz, Danyang Zhuo, and Yao Lu. 2023. Serving and Optimizing Machine Learning Workflows on Heterogeneous Infrastructures. In *PVLDB*.

[59] Ran Xu, Jinkyu Koo, Rakesh Kumar, Peter Bai, Subrata Mitra, Sasa Misailovic, and Saurabh Bagchi. 2018. VideoChef: Efficient Approximation for Streaming Video Processing Pipelines. In *ATC*.

[60] Shangliang Xu, Xinxin Wang, Wenyu Lv, Qinyao Chang, Cheng Cui, Kaipeng Deng, Guanzhong Wang, Qingqing Dang, Shengyu Wei, Yuning Du, and Baohua Lai. 2022. PP-YOLOE: An evolved version of YOLO. arXiv:2203.16250

[61] Tiantu Xu, Luis Materon Botelho, and Felix Xiaozhu Lin. 2019. VStore: A Data Store for Analytics on Large Videos. In *EuroSys*.

[62] Zhuangdi Xu, Gaurav Kakker, Joy Arulraj, and Umakishore Ramachandran. 2022. EVA: A Symbolic Approach to Accelerating Exploratory Video Analytics with Materialized Views. In *SIGMOD*.

[63] Zhihui Yang, Zuozhi Wang, Yicong Huang, Yao Lu, Chen Li, and X. Sean Wang. 2022. Optimizing Machine Learning Inference Queries with Correlative Proxy Models. In *PVLDB*.

[64] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets.. In *HotCloud*.

[65] 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*.