

Mixer: Efficiently Understanding and Retrieving Visual Content at Web-scale

An Qin
Baidu, Inc.
qinan@baidu.com

Mengbai Xiao
School of Computer Sci. & Tech.
Shandong University
xiaomb@sdu.edu.cn

Yongwei Wu
Baidu, Inc.
wuyongwei03@baidu.com

Xinjie Huang
Baidu, Inc.
huangxinjie@baidu.com

Xiaodong Zhang
Department of Computer Sci. & Eng.
The Ohio State University
zhang@cse.ohio-state.edu

ABSTRACT

Visual contents, including images and videos, are dominant on the Internet today. The conventional search engine is mainly designed for textual documents, which must be extended to process and manage increasingly high volumes of visual data objects.

In this paper, we present Mixer, an effective system to identify and analyze visual contents and to extract their features for data retrievals, aiming at addressing two critical issues: (1) efficiently and timely understanding visual contents, (2) retrieving them at high precision and recall rates without impairing the performance. In Mixer, the visual objects are categorized into different classes, each of which has representative visual features. Subsystems for model production and model execution are developed. Two retrieval layers are designed and implemented for images and videos, respectively. In this way, we are able to perform aggregation retrievals of the two types in efficient ways. The experiments with Baidu's production workloads and systems show that Mixer halves the model production time and raises the feature production throughput by 9.14x. Mixer also achieves the precision and recall of video retrievals at 95% and 97%, respectively. Mixer has been in its daily operations, which makes the search engine highly scalable for visual contents at a low cost. Having observed productivity improvement of upper-level applications in the search engine, we believe our system framework would generally benefit other data processing applications.

PVLDB Reference Format:

An Qin, Mengbai Xiao, Yongwei Wu, Xinjie Huang, and Xiaodong Zhang. Mixer: Efficiently Understanding and Retrieving Visual Content at Web-scale. PVLDB, 14(12): 2906-2917, 2021.
doi:10.14778/3476311.3476371

Qin and Xiao contributed equally to this work as the first author.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 14, No. 12 ISSN 2150-8097.
doi:10.14778/3476311.3476371

1 INTRODUCTION

As cell phones become necessities in the human society, the growth of user-generated contents (UGC) [17] in the format of texts, images and videos in the Internet are explosive. It is a common practice for one to take pictures or record videos and then to share their living experiences via online social networks (OSNs) from time to times. Haokan¹, Douyin², and Xigua³, the most popular Internet content providers in China, reported that their daily active users (DAU) have reached to 100-million. Within 2018 in China, more than 4.8 billion user-created videos are distributed and watched over the Internet, while this number was only 1.7 billions in 2017 [38]. Data management at Web-scale refers to provide fast and high quality service to users' data access requests in the Internet, where the growth rate of request volumes and data volumes are explosive. We have designed and implemented a data management system in such an Internet environment, focusing on efficiently understanding and fast retrieving visual data objects at web-scale by addressing the following two major challenges.

1. The challenge of achieving the effectiveness of machine learning in production systems. With the fast advancement of deep learning (DL) techniques, understanding visual objects of images and videos by machines has become feasible. In 2015, convolutional neural networks (CNNs) was demonstrated a better accuracy and reliability than that of human experts in image classification for the first time [18]. However, incorporating these component-wise accomplishments in a comprehensive production environment is non-trivial. The rapid evolution and wide applications of DL research give us both opportunities and challenges, where novel models are being developed and existing models are frequently updated to respond the high dynamics of various applications. Unlike traditional data management system that mainly produces textual features, the system in our design produces visual features frequently with the support of efficient model productions. It is highly desirable in practice to create an interactive tool for software engineers to quickly test various DL models, and timely publish the best selected models. This process is called model production. In addition, DL models are specially designed for diverse understanding tasks, such as a model for human face recognition [11], and another

¹<https://haokan.baidu.com/>

²<https://www.douyin.com/>

³<https://www.ixigua.com/>

model for scoring the visual clarity [44, 47]. These models are usually computational-intensive, and thus to thoroughly understand the explosively growing volumes of visual objects is unbearable. Performance factors, such as high accuracy and high throughput are important considerations in the algorithm design for producing visual features. Computing task scheduling also faces unique challenges in such a complex data management system. Moreover, the lack of fine-grained virtualization and the unstable resource provision in GPU is another concern for model acceleration in an efficient way.

2. The challenge of gaining high performance of visual data retrievals. Retrieving similar visual objects is a major function in the system. The image-to-image search can be realized by retrieving the feature vector of the query with approximate nearest neighbor search (ANNS) [9, 23, 49] in a database. In such a system, efficient index building for the real-time visual data objects is sophisticated and challenging. Most studies on the video-to-video search focus on near-duplicate video retrieval (NDVR) [22, 28, 29, 41]. However, NDVR only supports retrieving videos that are highly similar or duplicated to the query. The recall rate declines significantly by extending NDVR to general cases. In addition, people edit a video and redistribute it today, leading to a large amount of *edited videos* all over the Internet. Some of the edited videos are semantically similar to the original one, like adding simple visual effects to a video, while some are not, like duplicating frames from a scenery video just as the background of a talk show video. Furthermore, some videos are maliciously edited and should not be considered as the retrieval results at all, like concatenating a small video clip repeatedly. Thus, recognizing the having-edited videos is another key to improving the video retrieval quality. Furthermore, properly ranking a visual object appeared in the retrieval results is important and nontrivial. A video or an image is usually uploaded to multiple platforms by its author for a reason of gaining profits. It is cost-effective to evaluate these duplicated or nearly-duplicated objects as a group and determine their ranks. The associate resources on the platforms, such as *Likes* and *Comments*, is helpful in assessing visual objects. However, recursively fetching similar objects for every entity in the retrieval result and then aggregating their features are computationally expensive, which may lead to high execution latencies. Therefore, how to efficiently perform the aggregation retrieval is critical for a highly responsive search engine.

To address the above-mentioned issues, in this paper, we present Mixer, a system to efficiently understand and quickly retrieve visual data objects, as an essential system infrastructure in the Baidu search engine. Mixer categorizes the crawled visual objects by a CNN-based classifier [31, 32, 46] into classes and produces the visual features accordingly. The features are stored in a unified container, called compound feature vector (CFV), as the forward indices. Mixer also includes a comprehensive model production system tool that assists engineers in developing, testing, and publishing DL models timely. To efficiently execute the models, Mixer abstracts a model as a directed acyclic graph (DAG), where a vertex is a computation unit or an operator to be individually encapsulated into a container, and the edges guide how the visual data objects flow among models. By launching and revoking containers, executing models can scale easily at the operator level. In addition, a heuristic scheduling algorithm is developed to adjust resource provision to the operators for

maximizing the feature production throughput. For the retrieval of images, the feature vectors served as the inverted indices, are encoded with product quantization (PQ) [24] and are organized in a GNOIMI-like architecture [53], so that they could be fetched with ANNS approaches. For the retrieval of videos, Mixer extracts key-frames via entropy analysis [50] and collects all similar frames by leveraging the frame-level features. Similar videos containing these frames are hence located, and the candidate list is further refined by filtering out the edited ones. The objects appeared in the retrieval results are aggregated with their similar objects and associate resources to ensure an accurate ranking. In order to keep the system to be responsive, the visual objects are organized as a graph structure, where the edges, i.e., the similarities, are sorted in a flat table. In this way, neighbors or similar objects could be retrieved by using a range scan. A breadth-first search is used to discover the similar objects, and their features are aggregated before the ranking. We evaluate the model production and the model execution based on Baidu's production workloads. The results show that Mixer halves the execution time for producing the models, and the scheduling algorithm improves the feature production throughput by 9.14x. Mixer also achieves the precision and recall rate of video retrievals at 95% and 97%, respectively. Mixer makes the search engine highly scalable for increasingly large volumes of visual data objects at a low cost. Having observed productivity improvement of upper-level applications in the search engine, we believe our system framework would generally benefit other related data processing applications.

Our contributions are summarized as follows:

We have designed and implemented Mixer, a system to efficiently understand and retrieve visual data objects at web-scale, which is a key component in the Baidu search engine. We show that the performance of understanding visual objects is determined by both the feature production and the model production. Consequently we implement two individual subsystems for producing models and executing models to gain high performance for Mixer.

We design a video retrieval framework that is able to recognize edited videos. We additionally develop the aggregation retrieval function that combines multiple rounds of search results to improve the ranking quality in a responsive way. We evaluate Mixer with production workloads and show the effectiveness of our design in processing visual data objects at web-scale.

The rest of the paper is organized as follows. Section 2 introduces the background of processing visual data in Baidu. We discuss the visual content understanding in Section 3 and the visual content retrieval in Section 4. The performance of Mixer is evaluated in Section 5. Section 6 presents related work and Section 7 concludes the work.

2 BACKGROUND

2.1 Visual Resources

In Baidu, we keep crawling hundreds of millions of *textual resources* and *visual resources* from the Internet on a daily basis. A textual source could be an article or a collection of comments, while visual

Figure 1: Typical webpages nowadays

sources are data objects like images and videos. The typical webpages crawled from the Internet are shown in Figure 1, where the visual objects could be a supplement to the article, such as the left one for news published at *cnn.com*, or vice versa, such as the right one for a video uploaded to *youtube.com*. The visual objects have distinct Uniform Resource Locators (URLs) from that of the webpage containing them⁴, so they are crawled as individual resources in Baidu. It is worth noting that a video is usually accompanied with a cover image selected by the author, which has a separate URL from the video. The correlations between these resources are also captured. In Baidu, a video is stored with two fields: *furls* contains the player pages and *cimg* refers to the cover image. An image also has a field of *furls* indicating the pages containing the image.

2.2 Mixer in Baidu

Mixer is placed between Baidu infrastructure systems and various applications, being independent of the textual webpage processing pipeline. The infrastructure systems support Mixer with major services, such as local storage systems, accessing to the archive [36], and others. When a webpage is fetched from the Internet by the *crawler*, it is dismantled into textual documents and visual objects, i.e. images and videos. While the textual documents are processed in the conventional pipeline [37], which involves in indexing by *indexer* and ranking by *ranker*, and the visual objects are delivered to Mixer for special processing. Part of the indexing and ranking functions in Mixer are implemented themselves while the others are provided by the indexer and ranker. Mixer supports the upper-level applications with APIs that can be used to realize complex logic. Figure 2 details the software stack in Baidu, which includes Mixer. **The predecessor system of Mixer:** Before Mixer, Baidu understands visual objects following the MapReduce model [10], where the images and videos are processed in batch. But with this method, the feature production becomes the bottleneck of the whole system soon. Before the development of Mixer, it takes 1.5 months to generate city tags for 2 billion videos, while the amounts of newly crawled videos and images quickly reach 40 millions/day and 20 millions/day, respectively. Another problem is the image/video retrieval. Before Mixer, the videos/images are associated with the textual tags that are either collected from the documents of the *furls* or extracted from directly understanding the visual content

⁴While the article shares the same URL with the webpage, the comment section usually has a different URL.

Figure 2: Mixer in Baidu's software stack

Figure 3: Content understanding in Mixer. The processing of visual objects used to query or to index differs on two steps. A visual object is deduplicated and its CFV is stored into the forward index table only if it is crawled from the Internet for indexing.

with DL models. However, the tags are not descriptive enough to help retrieving visual objects at high precision and recall.

3 CONTENT UNDERSTANDING

The textual content has been well processed in Baidu [37]; the search engine should also be extended to handle the visual resources. To efficiently process the visual objects crawled from the Internet, we deploy a classifier in Mixer, which categorizes an input image or video into one of 12 classes. Despite that different features are produced for different classes, the features of visual objects are held in a unified container. To efficiently craft the visual features, on one hand, we have designed and integrated a model production subsystem into Mixer for fast model development and update, and, on the other hand, we have carefully optimized the model execution, aiming to maximize the feature production throughput.

3.1 Visual Data Object Processing

In content understanding, Mixer produces a *compound feature vector* (CFV) for a visual object. The CFV is then stored in the *forward index table*, whose key is the *resource ID* of the visual object. When the resources or the models are updated, the features should be

Table 1: Representative features in the CFV

textual features	<i>furls, cimg, likes, resolution, duration</i>
visual features	<i>SIFT [33], clarity [44, 47], celebrities</i>
miscs	<i>d-indicator, layout</i>
<i>d-indicator: Indicator of a dead resource</i>	

re-crafted accordingly. Building such a forward index table helps update the existing features efficiently, which is critical in improving the overall retrieval performance. Different features in the CFVs are further built into different *inverted index tables* for effective retrievals. Figure 3 overviews the basic process of Mixer crafting the CFVs from visual resources.

3.1.1 Coarse-Grained Deduplication. Copies of images and videos are massively distributed across the Internet. Since the extraction process of visual features is highly computation-intensive, effectively eliminating the duplicates is a key to improving the overall performance. When deduplicating a crawled visual object, we remove its watermark region if it has any, and employ a hash-based algorithm to generate a fingerprint for it [30]. If the same fingerprint exists already, the video or image is considered to be duplicate, thus it is discarded. All these steps are integrated in the *coarse-grained deduplication module*. With this deduplication effort, Mixer removes 50% duplicate images and 25% duplicate videos before feature crafting.

3.1.2 Compound Feature Vector. A CFV is a 128-dimension vector, which consists of three types of features, namely *textual features*, *visual features* and *miscs*. The textual features are extracted from the visual object without visual understanding, such as the resolution of an image or the playback duration of a video. A feature is a visual feature if it is directly generated from the visual content. Otherwise, this feature is attributed to the miscs. Table 1 gives representative features in the CFV, where some features are exclusively built for images and some are for videos. The features in the CFV are one of the following types: *boolean*, *string*, *numeric*, *tag*, and *vector*. For example, *SIFT* [33] is a vector and *celebrities* is a set of tags that are people names. In Mixer, the *resource ID (rid)* is used as the key for the CFV that uniquely identifies a visual resource in the forward index table.

3.1.3 Feature Crafting. Some features can be generated trivially, such as the *furl*; but many others are not, which are particularly the visual ones. Since the visual resources on the Internet are rich and diverse, it is impossible to find an understanding model for all. In practice, different models are developed to craft various visual features. However, understanding the visual objects with all models is computationally unbearable. Thus, Mixer employs a CNN-based classifier [31, 32, 46] to categorize all crawled images and videos into 12 major classes, including *scenery*, *movie*, *portrait*, and others. For different classes of visual contents, different features are crafted. With the evolution of the system, we notice that the bottlenecks of feature production come from two sources: the model production and the model execution.

Figure 4: The model production and execution are processed in the following steps. 1) The developers publish their models to a pool; 2) Online tests are carried out to generate the performance profile for the uploaded models; 3) The scheduler master encapsulates the operators into containers and deploys them onto computation nodes according to the performance profile and the resource monitor; 4) The traffic manager directs the data flow to the nodes; 5) Containers are launched/revoked based on the dynamic resource provision.

3.2 Model Production and Execution

Most visual features are produced by advanced DL models. In order to incorporate the state-of-the-art development from the machine learning community, Mixer consists of a comprehensive model production subsystem, which allows the developers to upload and test their models in an agile and responsive way. These models are managed in a pool so that Mixer could deploy them in the production environment on demand, allowing fast iterations of the system. When executing the models, our major concern is how to make Mixer best utilize the computational resources. Figure 4 shows an overview of the model production and execution.

3.2.1 Model Production. A *feature production model (FPM)* is one entity producing one type of feature in Mixer. In Figure 4, a model production example of the FPM is presented. The FPM accepts a video as the input and outputs a boolean value telling if a logo is contained in it. Inside the FPM, three operators are concatenated, which are *sampler* [50] that decodes and samples frames from the input video, *OCR* [13] that recognizes texts in the frames, and *LOGO* [39] that identifies the commercial logos. The operators are either executed on CPU or on GPU. The FPM is thus abstracted as a directed acyclic graph (DAG), where the vertices are operators and the edges are the data flows between them. It is worth noting that an operator could be an FPM itself.

After a developer uploads her model, she can request Mixer to deploy and test her model by using a small portion of online traffic. The experimental results are returned to the developer so that she can tune the parameters and further refine the model. Mixer also supports developers with tools that use AutoML [20, 40, 54] to adjust parameters for optimization or automatically turn the GPU code

Figure 5: The number of published models and the training time at 80th percentile before and after the model production subsystem goes online.

Figure 6: The common operators in two FPMs could be merged into one if the input data are the same.

into TensorRT-supported ones to gain high performance [2]. Before the final publishing in the *FPM pool*, we will carry out thorough tests towards the models and generate the performance profiles for proper model deployment in the production environment.

Figure 5 compares the model production efficiency before and after the model production subsystem goes online. We measure the number of published models every three months, and it shows that the increasing rate is improved by 2.31x, from 6 models/month to 13.83 models/month. To further confirm if such improvement results really come from the our efforts to raise the efficiency of model development, we also collect the time used to train models by the developers right before and after the subsystem goes online. We find that the iteration time of model training at 80th percentile decreases from 593 minutes to 295 minutes, which closely matches the improvement observed from increasing model number.

3.2.2 Model Execution. When deploying and executing an FPM, Mixer abstracts it as a DAG, schedules the operators, and directs traffic according to the edges. An overview of this process is presented at the right side of Figure 4. A vertex, i.e., an operator, in the DAG is mapped to a group of identical container instances. By checking the performance profile and consulting with the resource manager module, the scheduler deploys the instances onto physical nodes properly. Then the traffic manager module is informed to direct traffic to these nodes. During the execution, container instances are launched or revoked to respond to a dynamic resource provision, and the traffic is redirected accordingly.

Before the execution, we find an optimization opportunity to merge the same vertices from different DAGs if the processed data are the same. For example, to detect the advertisement and the

Algorithm 1 The heuristic algorithm arranging computing resources to the operators in an FPM

```

1: procedure S (D, R)
2:   Input: D: the DAG of the FPM; R: the available computation resources; r: the minimum resources to be scheduled; updateActiveNodes(): mark active nodes in the DAG; activeSinks(): the active sinks of the DAG; measureThrpt(): measure the out throughput of an operator; devoteRes()/revokeRes(): devote/revoke resources from an operator
3:
4:   updateActiveNodes(D)
5:   S ← activeSinks(D)
6:   while S != ∅ do
7:     for s in S do
8:        $T_0 \leftarrow \text{measureThrpt}(s)$ 
9:       devoteRes(s, r)
10:       $T \leftarrow \text{measureThrpt}(s)$ 
11:      if  $T > T_0$  then
12:        revokeRes(s, r)
13:        schedule(D - s, R)
14:      else
15:         $R \leftarrow R - r$ 
16:        updateActiveNodes(D)
17:      S ← activeSinks(D)

```

commercial logo in a video, we both need sample frames and recognize texts from them. Figure 6 illustrates such a case. Furthermore, for the operators that are widely used as the essential building blocks across FPMs, such as the sampler that is almost used by every video-related FPM for preprocessing, they are turned into resident services from which the resources are never revoked.

The resource provision is highly dynamic during the model execution. New resources might be available and the arranged resources could also be claimed back by other services with higher priorities. In addition, the optimal resource distribution to the operators in an FPM varies along the execution lifetime. It is a waste to arrange any resource to the last operator(s) in the DAG at the beginning stage of the execution since no data has arrived. Another constraint in the resource provision is that the new resources sometimes are too scarce to be allocated to the operators evenly. Thus we develop a heuristic scheduling algorithm in Mixer to dynamically allocate computing resources among operators in an FPM. Algorithm 1 describes how the resources are devoted or revoked in Mixer with the objective of maximizing the processing throughput of an FPM. In the algorithm, we gradually activate nodes in a DAG from the source to the sink according to the data arrival. Among the activated nodes, we allocate the resources in the reverse direction: the resources are experimentally devoted into the sinks of the activated sub-graph for throughput improvement observations. If there is no improvement observed, the resources are revoked and we recursively schedule the nodes in a smaller sub-graph by eliminating the sinks.

Another challenge of efficiently executing models is to implement and manage a multi-tenant mode on GPUs. Unlike CPU, GPU

Figure 7: In the IR layer, the second-level centroids point to two lists, namely the *base inverted indices* contained in continuous memory pages and the *real-time inverted indices* as a link list. To build an inverted index for an input vector v on-the-fly, the new vector is 1) encoded and written into a WAL. Then, 2) the closest centroid with respect to the Euclidean distance is located, and the address of the new vector is inserted as the head of the real-time inverted indices. 3) Following the commits of the WAL, the real-time inverted indices are periodically merged into the base inverted indices. 4) For searching the closest neighbors of a query vector q , all neighboring centroids are traversed and a top-k list is generated based on the comparison results.

cannot be virtualized in fine-grain [45]. A container only agrees a complete GPU card to be bound. As a result, a physical node equipped with four GPU cards (the most deployed setup in our data centers) can only tolerate four instances of GPU operators. Running additional instances leads to potential resource contention and, more severely, system crash. However, running only one instance on a GPU card significantly underutilize the computing resources. To dynamically utilize available computing resources on GPU, we test if any two FPMs could effectively run on a single GPU card before the model publishes and records this in the performance profile. With this updated information, more instances of GPU operators are allowed to execute on a physical node, thus, we eventually enhance the overall execution capability of Mixer.

4 CONTENT RETRIEVAL

The features generated in content understanding in types of *tag* and *vector* are either built into the inverted indices or used as a search query. Since building inverted indices for tags follows the conventional techniques for textual webpages [37], we focus on organizing the vectors. In Mixer, currently there are two vector-typed features, the SIFT feature and the facial feature [11]. In order to effectively retrieve visual objects with these features, we make Mixer handle images and videos differently. For the images, we build an Image Retrieval Layer (IR layer) that accepts a feature vector, and returns a list of similar images if the input is a query

Figure 8: We configure the *Video Retrieval Layer* (VR layer) to help us retrieve duplicate videos, where the information entropy analysis is setup for feature extraction and the temporal order verification as well as the similarity score calculation are used for candidates refinement. The sequence to search duplicate videos upon a query is in the following steps: 1) A set of key-frames are extracted from the query video and their SIFT features are calculated; 2) The features are used to search the most similar frames in the IR layer; 3) By aggregating these frames, a candidate video list is formed; 4) The candidate videos are further refined by *temporal order verification* so that only the videos with a consistent frame order to the query video persist; 5) The similarity scores of the videos appeared in the final candidate list are further calculated.

vector, or inserts the input into the inverted indices otherwise. To compare the videos, we design a Video Retrieval Layer (VR layer), whose query could be a video and returns the similar videos by leveraging the IR layer.

4.1 Image Retrieval Layer

The IR layer is a GNOIMI-like [53] architecture where the feature vectors are encoded with Product Quantization (PQ) [24]. Compressing a 1024-byte SIFT feature into a 128-byte PQ vector improves the throughput by 3-4x. To cluster the vectors, two levels of centroids are trained and there are K centroids in each level. Empirically, K is set to 256 in Mixer to avoid too large subcells or too many neighbors in the search space. A write ahead log (WAL) is incorporated to accept newly crafted feature vectors, and the vectors are merged into the inverted index files periodically. Figure 7 illustrates the structure of the IR layer, and how it handles the indexing and searching operations.

4.2 Video Retrieval Layer

The VR layer is a video-to-video retrieval framework, where an image is also a legal query and is processed as a video with only one frame. In our production environment, varying goals are expected to be achieved, such as finding duplicate videos or searching specific people in videos. Thus, we generalize the VR layer that follows

Figure 9: Verifying the temporal consistency between the input video and a candidate video with the frame mapping. (a) shows that the playback order of the candidate video conforms to the input video. In (b), the candidate video has an inconsistent playback order. (c) presents a case that the candidate video duplicates frames from the input video.

several steps to process the query video. 1) In *feature extraction*, the query video is pre-processed and visual features are extracted. 2) These features are then sent to the IR layer for searching frames that contains similar features. 3) Using the similar frames to form the candidate video list, we still need to eliminate part of the list for *candidate re-nement*. 4) Finally we carry out the *result ranking* to eventually return the retrieval result.

Within the processing pipeline, feature extraction, candidate re-nement, and result ranking are configured with different modules for varying video retrieval tasks. So far we have configured the VR layer to provide the functions of searching duplicate videos, searching similar videos that are recorded for the same incident, recognizing people in videos, and searching 360-degree videos. Next we will use the configuration of searching duplicate videos as a representative case to illustrate the VR layer in details.

Figure 8 presents the workflow of the VR layer that is configured to search duplicate videos. In the feature extraction, information entropy analysis [50] is employed to extract key-frames and their SIFT features are calculated, which are used as the input to the IR layer for searching similar frames. If the VR layer is configured to recognize people, human faces and their features in the query are the extraction target. To process 360-degree videos, the query video is transformed into cubic projection and is then cut into six individual frames before the feature extraction.

As the candidate video list is formed based on the similar frames returned from the IR layer, this method also qualifies a number of *edited videos*. Some of them are maliciously edited, such as to concatenate a small video clip repeatedly, and some are completely different contents at the video level, e.g. a personal talk shows duplicating frames from scenery videos as the background. These edited videos should be detected and filtered out from the candidate video list. So we adopt the *temporal order verification* in candidate re-nement to remove the edited videos efficiently. Particularly, for a candidate video with n frames, we can generate a *frame mapping* f_1, \dots, f_n representing the frame correlation between it and the input video, where I_k contains the IDs of the similar frames in the input video. To detect the inconsistent playback order in the

Figure 10: The graph structure representing the resource relationship. A video v_1 is embedded in multiple player pages d_1, d_2, \dots , and it uses p_1 as its cover image. Before generating the aggregation CFV of v_1 , the features of related resources are collected. The associate resources are located by *furls* and *cimg* in the CFV of v_1 , and the comparable resources, i.e., v_2, v_3, p_2 , and p_3 are retrieved via two range scans. For these comparable resources, we still collect the associate resources of them, but exclude their comparable resources.

candidate video, we can check the rule as follows:

$$9i \in I_a \ 9j \in I_b \ i > j \wedge a < b:$$

Or a candidate video with duplicate frames could be recognized with the rule:

$$9i \in I_a \ 9j \in I_b \ i = j \wedge a \neq b:$$

The example videos following the rules are shown in Figure 9. More importantly, new rules could be installed with little efforts, and this helps extend the temporal order verification module against emerging types of videos in the future.

Accompanied with the final video candidates, their similarity scores to the query video are also generated based on the frame-level similarities and the frame mapping. The VR layer calculates the similarity score of a candidate video follows the formula:

$$\frac{1}{n} \sum_{a=1}^n \frac{\sum_{i \in I_a} s_{ai}}{|I_a|} \cdot \frac{\sum_{j \in I_b} |I_j|}{|I_a| |I_b|} \cdot \frac{1}{|I_a| |I_b|} \cdot \sum_{i,j \in I_a, I_b} \frac{1}{|I_a| |I_b|};$$

where s_{ai} is the similarity score between the a th frame of the query video and the i th frame of the candidate video, and I_b is the next non-empty frame set from I_a . The formula shows the similarity score between videos are from two major components, one comes from the frame-level similarities and the other is the distance difference between mapped frames. We use two coefficients α_a and α_b to weigh the two components, which are frame-dependent. In practice, the first frame and the last frame have higher weights than others.

4.3 Aggregation Retrieval

To comprehensively assess a visual resource retrieved from the IR/VR layer, we need to consider all relevant resources, such as a video encoded in different resolutions and the player pages with helpful textual information at various platforms. The process of locating and grouping the relevant resources for the retrieved result

Figure 11: The *City Names* FPM. OCR [13] and ASR [8, 35] are used to extract textual information from the frames and the audio, respectively. BERT [12] is a pre-trained model for language understanding, and selector is merely a classifier choosing the cities from over 350 candidates with k-means.

is named *aggregation retrieval*. We generate an *aggregation CFV* based on the CFVs of all resources in a group, which will be used to determine the rank of the visual resource shown to the user.

In addition to the associate resources that are explicitly presented and updated in the CFV, e.g., the *furl* and the *cimg* (see Section 3.1.2), we also need to collect information from the *comparable resources* for generating the aggregation CFV. The comparable resources are the similar objects fetched from the IR layer or the VR layer. However, recursively retrieving the visual objects significantly increases the response latency. Thus, the comparable resources in Mixer are organized as graphs where the edges are weighed by the similarity scores between them. The edge is directed due to the fact that two images might have different similarity scores in each other's similar object list. The edges are calculated when the visual objects are built into the inverted indices in the IR/VR layer, and are stored in the *similarity table*. The edges are sorted by their sources so that the comparable resources of a visual object could be fetched efficiently by a range scan. Figure 10 shows how the visual resources are abstracted as graphs and how the edges are hold in the similarity table.

To discover the comparable resources that should be grouped, Mixer conducts a breadth-first-search in the graph and adds the sink into the group if the edge weight is high enough. For a vertex with two or more hops from the visual object, the similarity score is the product of all edges along the path. The grouping algorithm stops if enough vertices, which is empirically set as 10, have been selected or no sink with an enough score at the current hop is found. Figure 10 illustrates an example of grouping the comparable resources v_2 and v_3 for a video v_1 and p_2 and p_3 for its cover image p_1 . The fields of the aggregation CFV are synthesized in different strategies, e.g., we accumulate the *likes* and select the maximum to represent the *clarity*.

5 EVALUATION

All of our experiments are conducted on a K8s-like [1] PaaS (Platform as a Service) with 2500 machines in a cluster. Each machine is equipped with an Intel Xeon Gold 6271C processor (24 cores at 2.6 GHz), 32G DDR4-2666 main memory, four NVIDIA T4 GPU cards, and a 4T NVMe-SSD. The CPU on a machine is abstracted into 1500 standard cores in resource provision. The machines are connected through 25 Gbps full-duplex Ethernet.

Table 2: Performance of *City Names* using different scheduling algorithms

	heuristic alg.	xed-quota alg.
<i>time elapsed (hours)</i>	10	88
<i>avg. throughput (FPS)</i>	1481	162

5.1 Scheduling Performance

We first evaluate the heuristic scheduling algorithm proposed to optimize the resource provision in each FPM (Algorithm 1). The experiments are carried out with the *City Names* FPM that recognizes cities from an input video. The *City Names* FPM is presented in Figure 11. We select this FPM since all of its operators can be both scheduled on CPU and GPU. In the experiments, a set of 160K videos are the input to be processed, and the total resource provision is capped. There are 9000 standard CPU cores and 2650 GPU cards to be allocated. As the baseline, we use a xed-quota scheduling algorithm that arranges the computation resources according to the throughput ratio of operators when using the same hardware resource. Specifically, {5000, 1000, 2500, 500} standard CPU cores and {1000, 500, 1000, 150} GPU cards are allocated to {OCR, ASR, BERT, selector} in the xed-quota scheduling. The overall performance of our heuristic algorithm and the baseline are presented in Table 2, where our heuristic scheduling algorithm outperforms the xed-quota scheduling by 7.4x and 9.16x in terms of the overall processing time and the average throughput, respectively. We also look into the scheduling details by profiling the CPU/GPU resource provision whenever the algorithm is activated. The scheduling algorithm is setup to work every two hours to avoid high overhead raised by frequent resource reallocation. The profiling results are shown in Figure 12, in which we use *JOB1* to represent the heuristic algorithm and use *JOB2* as the baseline. The y-axis is the resource provision in standard CPU cores/GPU cards. From the figure, we can see that in the early stages (Stages 1 and 2), the computing resources are mostly devoted into OCR and ASR since the data have just arrived. When the execution is close to the end, most computing resources are revoked from OCR and ASR and are arranged to BERT and selector. Such a migration significantly optimizes resource provision compared to the xed-quota scheduling algorithm.

To evaluate the heuristic scheduling algorithm more comprehensively, we measure the GPU utilization in our data centers before and after the scheduling mechanism is used. Other metrics can hardly distinguish the effects of the heuristic scheduling: The CPU standard cores are always fully allocated; the feature production is workload-dependent so its throughput does not necessarily vary accordingly. The results are shown in Figure 13, in which the y-axis is the GPU utilization and the x-axis is the timeline before and after installing the heuristic algorithm. We at first apply the heuristic algorithm over 5% of the overall workloads, and this effectively improves the GPU utilization to 15.4% from 12.1% in last month. While 30% workloads are scheduled with the heuristic algorithm, the GPU utilization is further increased to 18.5% at that month. To the end, the GPU utilization grows to 24.8% as all workloads are scheduled with the heuristic algorithm. As a comparison, the GPU utilization was only 11.2% when the xed-quota scheduling algorithm was used.

Figure 12: The resource provision breakdown of different scheduling algorithms in *City Names*. *JOB1* adopts our heuristic scheduling algorithm, and *JOB2* adopts the fixed-quota scheduling algorithm.

Figure 13: The GPU utilization in the data centers before and after the heuristic algorithm is installed

Table 3: Performance of the VR layer

	precision@1	recall@1	bad-cases
similar video retrieval	94.7%	91.7%	n/a
duplicate video retrieval	97.6%	98.1%	0.14%

5.2 Retrieval Performance

Since the IR layer is built as a GNOIMI-like [53] architecture, which has been well-studied, we focus on the retrieval performance of the VR layer in this section.

5.2.1 VR Layer. We evaluate the VR layer in terms of retrieval performance and cost. The experiments are conducted based on the database that contains 1 billion videos, which are 5% of all videos

crawled from the Internet but serve 80% queries at that moment. Based on the VR layer, we configure and test two subsystems in our production environment by similar video retrievals and duplicate video retrievals. For similar video retrievals, Mixer fetches similar videos recorded towards an incident happening in the roughly same temporal-spatial space [28]. For duplicate video retrievals, we expect to extract the videos with the same content. In particular, these two subsystems share the same configurations in the VR layer except that rules are set differently in the temporal order verification module. For the duplicate video retrieval, rules for verifying temporal order are more restrict that only videos with roughly the same duration are qualified.

To evaluate the similar video retrieval, 1000 videos in the database are selected as the ground truth, and another 1000 videos are identified as similar ones by human. The similar videos are used as the query set. We measure two metrics, *precision@1* and *recall@1*, in the experiment. For each query video, the VR layer returns videos with their similar scores higher than 0.75 (this is an application-dependent value, see Section 4.2), and only the rank 1 video is selected as the result. We define *recall@1* as the number of the returned video divided by the cardinality of the query set since for some videos, their query results might be empty sets. *precision@1* is the portion of returned videos that are also in the ground truth set. The results are shown in Table 3, from which Mixer has achieved the recall at 91.7% by using the frame-level features and the precision at 94.7% with the assistance of the temporal order verification module.

For duplicate video retrievals, 1000 representative videos, which are selected by human to cover as many as possible types, are used

Figure 14: The cost of the VR layer vs. the database scaling

Table 4: Performance of aggregation retrieval (ar.)

	precision (%)		recall (%)	
	w/ ar.	w/o ar.	w/ ar.	w/o ar.
<i>plag. recognition</i>	84.3	93	n/a	n/a
<i>old news</i>	62.7	81.2	n/a	n/a
<i>origin lookup</i>	92.8	96.69	78.38	98.65

as the ground truth in the database. To generate the query set, we slightly edit these videos by attaching a logo onto their first frames. The query set consists of 1000 videos and we configure the similarity score threshold in the VR layer as 0.98. The *precision@1* and *recall@1* are measured and reported in Table 3, which are at 97.6% and 98.1%, respectively. Another important metric is *bad-cases*. Baidu constantly accumulates cases that impair user’s experience and strives to eliminate them in every development iteration. Before the VR layer is deployed, the videos are indexed and retrieved with tags. There are 1K bad-cases collected that fail to fetch the duplicate video from the database. These cases are also tested and the results are reported in Table 3. We found that only 0.14% of the bad-cases still fail the system, which shows the effectiveness of our design.

As the cost is always one of the most important concerns, we use cores/mv qps to define the cost of the VR layer, which measures how many standard cores we need to devote for maintaining 1 QPS to the database at the scale of 1 million videos (The resources for extracting key-frames and understanding visual content are not counted). For example, if we need to provision 10,000 standard cores for keeping 50 QPS to the database of 20 million videos, the cost is 10 cores/mv qps. We measure the overall cost of the duplicate video retrieval subsystem whenever 200 million additional videos are accumulated in the database. The results are shown in Figure 14, where the x-axis is the number of videos in the database and the y-axis is the cost. From this Figure, we see that the cost of our system decreases from 270 to 7.2 cores/mv qps while the video database scale increases from 200-million to 800-million. At this stage, we continue to make engineering efforts to reduce the cost of the VR layer. When the database scale increases from 800-million to 1-billion, the VR layer has been well developed and the cost converges at 7.2 cores/mv qps, showing the favorable scalability of our design.

5.2.2 Aggregation Retrieval. We evaluate the effectiveness of the aggregation retrieval module with three upper-level applications, i.e., *plagiarism recognition*, *old news*, and *origin lookup*. In *plagiarism recognition*, given a list of input videos, one should recognize any two of them have plagiarism. Application *old news* is implemented to identify if a news is out-of-date. Application *origin lookup* locates the origin video/image upon accepting a query. For the *plagiarism recognition* and *old news*, we use two query sets of 1K and 4K, respectively. The query sets are composed of problematic queries accumulated from users in Baidu. For the *origin lookup*, we randomly sample 10K user queries in the realistic workload. We report precision for all three applications but additional recall for the *origin lookup*. As the precision is calculated by recruiting people to manually identify the irrelevant entries, the ground truth used to calculate the recall is collected by sending the same query to the competitors, i.e., Google⁵, Sougou⁶, and Toutiao⁷, and combining all of the returned results. Table 4 shows the accuracy results of these applications with and without the aggregation retrieval. By precision measurements, plagiarism recognition, old news, and origin lookup each reaches 93%, 81.2%, and 96.69% after applying the aggregation retrieval, respectively. The improvements are 8.7% (from 84.3%), 18.5% (from 62.7%), and 3.8% (from 92.8%). Additionally, the recall in origin lookup is also increased by 20.27% (from 78.38% to 98.65%). Thus, we can observe that the aggregation retrieval can substantially improve the ranking performance of the upper-level applications.

6 RELATED WORK

6.1 Video Analytics at Scale

Recent advances in DL techniques have motivated a large body of studies in video analytics in large scales. Due to the high computational demands of DL models, processing all videos in the dataset might be unbearable so that efforts are made to narrow down a group of candidates that are actually sent to inference when a query arrives. NoScope [27] performs binary classification tasks over videos efficiently by adopting difference detectors and specialized networks as the filters before the reference network. Focus [19] also identifies the presence of a class of objects in videos. It indexes the videos with low cost CNNs at the ingestion time and uses the ground-truth CNN to verify the retrieval results. TAHOMA [3] introduces physical representations of the input image into the query optimization, e.g., carrying out resolution scaling or color-depth reduction, and with such techniques, the data handling cost could be drastically reduced. However, the above systems can hardly adapt to flexible user’s queries. To implement complicated queries, a SQL-like language, FRAMEQL, is developed in Blazelt [25, 26] to handle aggregation and limit operations over videos. Panorama [59] extends the system to handling unbounded vocabulary queries over videos. Another work [34] introduces probabilistic predicates that filter unqualified data blobs in the machine learning inference queries. VStore [52] identifies the conurbation of video formats as the central concern and explores an idea called backward derivation to realize the system. Vaas [6] features an

⁵<https://www.google.com/>

⁶<https://www.sougou.com/>

⁷<https://www.toutiao.com/>

interactive interface that is convenient for users exploring various methods for their analytics tasks.

6.2 Quantization-based ANNS

Another thread of research converts visual objects into tags or high-dimensional feature vectors in advance, and the retrieval is carried out by exactly matching the tags or finding vectors that are closed to the query. The quantization-based approximate nearest neighbor search (ANNS) is effective in the vector retrieval due to its efficiency [4, 5, 15, 21, 53]. As the GPU has been widely adopted in general data-processing systems [14, 55–58] due to its capacity of massively parallel computing, such a device could also accelerate the vector retrieval tasks. PQT [49] exploits the massive parallelism of GPU by processing multiple vectors simultaneously. Faiss [23] proposes a GPU k-selection algorithm, whose naive GPU implementation is the bottleneck of the processing pipeline. RobustIQ [9] further improves the GPU-based solution in terms of robustness. Due to the efficiency of processing user queries, the ANN search has been adopted in industrial systems [48].

6.3 Near-Duplicate Video Retrieval

Our video retrieval layer is inspired by the studies on issues of near-duplicate video retrieval (NDVR). Early approaches to recognizing near-duplicate videos employ hand-crafted features. A hierarchical solution [51] uses features from color histograms followed by expensive local features to identify the near-duplicate videos. Schemes based on Bag-of-Words [7, 41] are suggested to serve the increasing scale of videos. Another type of discriminate features could be generated via hashing algorithms [42, 43]. Recently, visual features extracted from the deep CNN architectures are demonstrated to be effective in NDVR tasks [16, 29]. In order to retrieve videos more than duplicates, NDVR is extended to fine-grained incident video retrieval (FIVR) [28].

7 CONCLUSION

In this paper, we present Mixer, an advanced search engine towards intelligently understanding and efficiently retrieving visual objects at web-scale. In Mixer, visual objects are categorized into representative classes; thus only a limited range of necessary features are extracted from a video or an image. Unlike the conventional search engine designed for textual documents, the feature production in Mixer must catch up the delivering pace of fast-evolving and diverse models, such as rapid advancement of deep learning algorithms. In order to incorporate the latest methods from AI community, we implement a full-edged model production subsystem in Mixer to help engineers develop, test, and publish their models systematically and conveniently. We also design a heuristic and generic algorithm to provision dynamic resources when executing various feature production models that are abstracted into DAGs. For the retrieval of visual objects in Mixer, an IR layer is developed to hold feature vectors extracted from images, and similar candidates are located via ANNS approaches. Another VR layer implemented on top of the IR layer can be configured to realize different video-to-video retrieval tasks. In addition, Mixer accurately ranks the retrieved visual objects by grouping the relevant objects in the aggregation retrieval module. Mixer has been deployed in daily operations for

three years in Baidu, demonstrating its values of cost-effectiveness, scalability and high-performance of the system.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their constructive comments. This work has been partially supported by the National Science Foundation under grants CCF-1513944, CCF-1629403, IIS-1718450, and CCF-2005884.

REFERENCES

- [1] 2020. Kubernetes. <https://kubernetes.io/>. Accessed Jul. 26 2021.
- [2] 2021. NVIDIA TensorRT. <https://developer.nvidia.com/tensorrt>. Accessed Jul. 26 2021.
- [3] Michael R. Anderson, Michael J. Cafarella, German Ros, and Thomas F. Wenisch. 2019. Physical Representation-Based Predicate Optimization for a Visual Analytics Database. In *Proceedings of the 35th IEEE International Conference on Data Engineering*. Macao, China, pp. 1466–1477.
- [4] Artem Babenko and Victor S. Lempitsky. 2015. The Inverted Multi-Index. *IEEE Transactions on Pattern Analysis and Machine Intelligence* vol. 37, no. 6 (June 2015), pp. 1247–1260.
- [5] Dmitry Baranchuk, Artem Babenko, and Yury Malkov. 2018. Revisiting the Inverted Indices for Billion-Scale Approximate Nearest Neighbors. In *Proceedings of the 15th European Conference on Computer Vision*. Munich, Germany, pp. 209–224.
- [6] Favyen Bastani, Oscar R. Moll, and Samuel Madden. 2020. Vaas: Video Analytics At Scale. *Proceedings of the VLDB Endowment* vol. 13, no. 12 (Aug. 2020), pp. 2877–2880.
- [7] Yang Cai, Linjun Yang, Wei Ping, Fei Wang, Tao Mei, Xian-Sheng Hua, and Shipeng Li. 2011. Million-scale Near-duplicate Video Retrieval System. In *Proceedings of the 19th International Conference on Multimedia*. Nice, France, pp. 837–838.
- [8] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. 2016. Listen, Attend and Spell: A Neural Network for Large Vocabulary Conversational Speech Recognition. In *Proceedings of the 41st IEEE International Conference on Acoustics, Speech and Signal Processing*. Shanghai, China, pp. 4960–4964.
- [9] Wei Chen, Jincai Chen, Fuhao Zou, Yuan-Fang Li, Ping Lu, and Wei Zhao. 2019. RobustIQ: A Robust ANN Search Method for Billion-Scale Similarity Search on GPUs. In *Proceedings of the 2019 International Conference on Multimedia Retrieval*. New York, NY, USA, pp. 132–140.
- [10] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation*. San Francisco, CA, USA, 137–150.
- [11] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. 2019. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. In *Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Long Beach, CA, USA, pp. 4685–4694.
- [12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. [arXiv:1810.04805](https://arxiv.org/abs/1810.04805) [cs.CL]
- [13] Yuning Du, Chenxia Li, Ruoyu Guo, Xiaoting Yin, Weiwei Liu, Jun Zhou, Yifan Bai, Zilin Yu, Yehua Yang, Qingqing Dang, and Haoshuang Wang. 2020. PP-OCR: A Practical Ultra Lightweight OCR System. [arXiv:2009.09941](https://arxiv.org/abs/2009.09941) [cs.CV]
- [14] Sofoklis Floratos, Mengbai Xiao, Hao Wang, Chengxin Guo, Yuan Yuan, Rubao Lee, and Xiaodong Zhang. 2021. NestGPU: Nested Query Processing on GPU. In *Proceedings of the 37th IEEE International Conference on Data Engineering*. Chania, Crete, Greece, pp. 1008–1019.
- [15] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized Product Quantization for Approximate Nearest Neighbor Search. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*. Portland, OR, USA, pp. 2946–2953.
- [16] Giorgos Kordopatis-Zilos and Symeon Papadopoulos and Ioannis Patras and Yiannis Kompatsiaris. 2017. Near-Duplicate Video Retrieval by Aggregating Intermediate CNN Layers. In *Proceedings of the 23rd International Conference on Multimedia Modeling*. Reykjavik, Iceland, pp. 251–263.
- [17] Lei Guo, Enhua Tan, Songqing Chen, Xiaodong Zhang, and Yihong (Eric) Zhao. 2009. Analyzing Patterns of User Content Generation in Online Social Networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Paris, France, pp. 369–378.
- [18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision*. Santiago, Chile, pp. 1026–1034.
- [19] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B. Gibbons, and Onur Mutlu. 2018.

- Focus: Querying Large Video Datasets with Low Latency and Low Cost. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation*. Carlsbad, CA, USA, pp. 269–286.
- [20] Frank Hutter, Lars Kottho, and Joaquin Vanschoren. 2019. *Automated Machine Learning: Methods, Systems, Challenges*. Springer Nature.
- [21] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. 2011. Searching in One Billion Vectors: Re-rank with Source Coding. In *Proceedings of the 36th IEEE International Conference on Acoustics, Speech, and Signal Processing*. Prague, Czech Republic, pp. 861–864.
- [22] Qing-Yuan Jiang, Yi He, Gen Li, Jian Lin, Lei Li, and Wu-Jun Li. 2019. SVD: A Large-Scale Short Video Dataset for Near-Duplicate Video Retrieval. In *Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision*. Seoul, Korea, 5280–5288.
- [23] Je Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-Scale Similarity Search with GPUs. arXiv:1702.08734 [cs.CV]
- [24] H. Jégou, M. Douze, and C. Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* vol. 33, no. 1 (Jan. 2011), pp. 117–128.
- [25] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. Blazelt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-based Video Analytics. *Proceedings of the VLDB Endowment* vol. 13, no. 4 (Dec. 2019), pp. 533–546.
- [26] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. Challenges and Opportunities in DNN-Based Video Analytics: A Demonstration of the Blazelt Video Query Engine. In *Proceedings of the 9th Biennial Conference on Innovative Data Systems Research*. Asilomar, CA, USA, online proceedings.
- [27] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Neural Network Queries over Video at Scale. *Proceedings of the VLDB Endowment* vol. 10, no. 11 (Aug. 2017), pp. 1586–1597.
- [28] Giorgos Kordopatis-Zilos, Symeon Papadopoulos, Ioannis Patras, and Ioannis Kompatsiaris. 2019. FIVR: Fine-Grained Incident Video Retrieval. *IEEE Transactions on Multimedia* vol. 21, no. 10 (Oct. 2019), pp. 2638–2652.
- [29] Giorgos Kordopatis-Zilos, Symeon Papadopoulos, Ioannis Patras, and Yiannis Kompatsiaris. 2017. Near-Duplicate Video Retrieval with Deep Metric Learning. In *Proceedings of the 2017 IEEE International Conference on Computer Vision Workshops*. Venice, Italy, pp. 347–356.
- [30] Kevin Lin, Hui-Fang Yang, Jen-Hao Hsiao, and Chu-Song Chen. 2015. Deep Learning of Binary Hash Codes for Fast Image Retrieval. In *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. Boston, MA, USA, pp. 27–35.
- [31] Xiang Long, Chuang Gan, Gerard De Melo, Xiao Liu, Yandong Li, Fu Li, and Shilei Wen. 2018. Multimodal Keyless Attention Fusion for Video Classification. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. New Orleans, LA, USA.
- [32] Xiang Long, Chuang Gan, Gerard De Melo, Jiajun Wu, Xiao Liu, and Shilei Wen. 2018. Attention Clusters: Purely Attention Based Local Feature Integration for Video Classification. In *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT, USA, 7834–7843.
- [33] David G Lowe. 1999. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the 7th IEEE International Conference on Computer Vision*. Kerkyra, Corfu, Greece, pp. 1150–1157.
- [34] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. 2018. Accelerating Machine Learning Inference with Probabilistic Predicates. In *Proceedings of the 2018 International Conference on Management of Data*. Houston, TX, USA, pp. 1493–1508.
- [35] Abdelrahman Mohamed, Dmytro Khokhlo, and Luke Zettlemoyer. 2020. Transformers with Convolutional Context for ASR. arXiv:1904.11660 [cs.CL]
- [36] An Qin, Dianming Hu, Jun Liu, Wenjun Yang, and Dai Tan. 2014. Fatman: Cost-saving and Reliable Archival Storage based on Volunteer Resources. *Proceedings of the VLDB Endowment* vol. 7, no. 13 (Aug. 2014), pp. 1748–1753.
- [37] An Qin, Mengbai Xiao, Jin Ma, Dai Tan, Rubao Lee, and Xiaodong Zhang. 2019. DirectLoad: A Fast Web-Scale Index System Across Large Regional Centers. In *Proceedings of the 35th IEEE International Conference on Data Engineering*. Macau, China, pp. 1790–1801.
- [38] QuestMobile. 2019. [Online]. QuestMobile Short Video Semi-annual Report 2019. <https://www.dx2025.com/wp-content/uploads/2020/02/QuestMobile-duan-shi-pin-2019-ban-nian-bao-gao.pdf>.
- [39] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. arXiv:1506.02640 [cs.CV]
- [40] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. arXiv:1703.03864 [stat.ML]
- [41] Lifeng Shang, Linjun Yang, Fei Wang, Kwok-Ping Chan, and Xian-Sheng Hua. 2010. Real-Time Large Scale Near-Duplicate Web Video Retrieval. In *Proceedings of the 18th ACM International Conference on Multimedia*. Firenze, Italy, pp. 531–540.
- [42] Jingkuan Song, Yi Yang, Zi Huang, Heng Tao Shen, and Richang Hong. 2011. Multiple Feature Hashing for Real-time Large Scale Near-duplicate Video Retrieval. In *Proceedings of the 19th International Conference on Multimedia*. Scottsdale, AZ, USA, pp. 423–432.
- [43] Jingkuan Song, Yi Yang, Zi Huang, Heng Tao Shen, and Jiebo Luo. 2013. Effective Multiple Feature Hashing for Large-Scale Near-Duplicate Video Retrieval. *IEEE Transactions on Multimedia* vol. 15, no. 8 (Dec. 2013), pp. 1997–2008.
- [44] Jianyi Wang, Xin Deng, Mai Xu, Congyong Chen, and Yuhang Song. 2020. Multi-level Wavelet-based Generative Adversarial Network for Perceptual Quality Enhancement of Compressed Video. In *Proceedings of the 16th European Conference on Computer Vision*. Glasgow, UK, pp. 405–421.
- [45] Kaibo Wang, Kai Zhang, Yuan Yuan, Siyuan Ma, Rubao Lee, Xiaoning Ding, and Xiaodong Zhang. 2014. Concurrent Analytical Query Processing with GPUs. *Proceedings of the VLDB Endowment* vol. 7, no. 11 (July 2014), pp. 1011–1022.
- [46] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. 2016. Temporal Segment Networks: Towards Good Practices for Deep Action Recognition. In *Proceedings of the 12th European Conference on Computer Vision*. Amsterdam, The Netherlands, pp. 20–36.
- [47] Xintao Wang, Kelvin C. K. Chan, Ke Yu, Chao Dong, and Chen Change Loy. 2019. EDVR: Video Restoration With Enhanced Deformable Convolutional Networks. In *Proceedings of the 2019 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. Long Beach, CA, USA, pp. 1954–1963.
- [48] Chuangxian Wei, B. Wu, S. Wang, Renjie Lou, C. Zhan, Feifei Li, and Yuanzhe Cai. 2020. AnalyticDB-V: A Hybrid Analytical Engine Towards Query Fusion for Structured and Unstructured Data. *Proceedings of the VLDB Endowment* vol. 13, no. 12 (Aug. 2020), pp. 3152–3165.
- [49] Patrick Wieschollek, Oliver Wang, Alexander Sorkine-Hornung, and Hendrik P. A. Lensch. 2016. Efficient Large-Scale Approximate Nearest Neighbor Search on the GPU. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*. Las Vegas, NV, USA, pp. 2027–2035.
- [50] Wenhao Wu, Dongliang He, Xiao Tan, Shifeng Chen, and Shilei Wen. 2019. Multi-Agent Reinforcement Learning Based Frame Sampling for Efficient Untrimmed Video Recognition. In *Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision*. Seoul, Korea, pp. 6221–6230.
- [51] Xiao Wu, Alexander G. Hauptmann, and Chong-Wah Ngo. 2007. Practical Elimination of Near-Duplicates from Web Video Search. In *Proceedings of the 15th International Conference on Multimedia*. Augsburg, Germany, pp. 218–227.
- [52] Tiantu Xu, Luis Materon Botelho, and Felix Xiaozhu Lin. 2019. VStore: A Data Store for Analytics on Large Videos. In *Proceedings of the 14th European Conference on Computer Systems*. Dresden, Germany, pp. 16:1–16:17.
- [53] Artem Babenko Yandex and Victor Lempitsky. 2016. Efficient Indexing of Billion-Scale Datasets of Deep Descriptors. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*. Las Vegas, NV, USA, pp. 2055–2063.
- [54] Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. 2019. Taking Human out of Learning Applications: A Survey on Automated Machine Learning. arXiv:1810.13306 [cs.AI]
- [55] Yuan Yuan, Rubao Lee, and Xiaodong Zhang. 2013. The Yin and Yang of Processing Data Warehousing Queries on GPU Devices. *Proceedings of the VLDB Endowment* vol. 6, no. 10 (Aug. 2013), pp. 817–828.
- [56] Kai Zhang, Kaibo Wang, Yuan Yuan, Lei Guo, Rubao Lee, and Xiaodong Zhang. 2015. Mega-KV: A Case for GPUs to Maximize the Throughput of in-Memory Key-Value Stores. *Proceedings of the VLDB Endowment* vol. 8, no. 11 (July 2015), pp. 1226–1237.
- [57] Simon Zhang, Mengbai Xiao, Chengxin Guo, Liang Geng, Hao Wang, and Xiaodong Zhang. 2019. HYPHA: A Framework Based on Separation of Parallelisms to Accelerate Persistent Homology Matrix Reduction. In *Proceedings of the 33rd ACM International Conference on Supercomputing*. Phoenix, AZ, USA, pp. 69–81.
- [58] Simon Zhang, Mengbai Xiao, and Hao Wang. 2020. GPU-Accelerated Computation of Vietoris-Rips Persistence Barcodes. In *Proceedings of the 36th International Symposium on Computational Geometry*. Zürich, Switzerland, pp. 70:1–70:17.
- [59] Yuhao Zhang and Arun Kumar. 2019. Panorama: A Data System for Unbounded Vocabulary Querying over Video. *Proceedings of the VLDB Endowment* vol. 13, no. 4 (Dec. 2019), pp. 477–491.