VOCALExplore: Pay-as-You-Go Video Data Exploration and Model Building

Maureen Daum
University of Washington
mdaum@cs.washington.edu

Enhao Zhang
University of Washington
enhaoz@cs.washington.edu

Dong He
University of Washington
donghe@cs.washington.edu

Stephen Mussmann
University of Washington
mussmann@cs.washington.edu

Brandon Haynes
Microsoft Gray Systems Lab
brandon.haynes@microsoft.com

Ranjay Krishna
University of Washington
ranjay@cs.washington.edu

Magdalena Balazinska
University of Washington
magda@cs.washington.edu

ABSTRACT

We introduce VOCALExplore, a system designed to support users in building domain-specific models over video datasets. VOCALExplore supports interactive labeling sessions and trains models using user-supplied labels. VOCALExplore maximizes model quality by automatically deciding how to select samples based on observed skew in the collected labels. It also selects the optimal video representations to use when training models by casting feature selection as a rising bandit problem. Finally, VOCALExplore implements optimizations to achieve low latency without sacrificing model performance. We demonstrate that VOCALExplore achieves close to the best possible model quality given candidate acquisition functions and feature extractors, and it does so with low visible latency (~1 second per iteration) and no expensive preprocessing.

PVLDB Reference Format:
Maureen Daum, Enhao Zhang, Dong He, Stephen Mussmann, Brandon Haynes, Ranjay Krishna, and Magdalena Balazinska. VOCALExplore: Pay-as-You-Go Video Data Exploration and Model Building. PVLDB, 16(13): 4188 - 4201, 2023.
doi:10.14778/3625054.3625057

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/uwdb/vocalexplore.

1 INTRODUCTION

Increasingly many scientific domains rely on video data, which is information dense and relatively easy to collect. Powerful libraries [5, 9, 16, 32] and data management systems [6, 7, 24, 33] exist to support users in storing and querying this video data. A key problem, however, is that those systems assume the user is already familiar with their data and, typically, already has one or more machine learning (ML) models to extract the desired information. In speaking with scientists at the University of Washington, we find that this is frequently not the case. Instead, scientists collect data *en masse*, but then struggle to explore it, understand it, build ML models for it, and finally use it to answer their scientific questions.

Consider, for example, a scientist who wishes to understand the behavior of animals in the wild using collars outfitted with cameras (we expand this example in Section 2.1). These cameras may easily produce terabytes of data. Because scientists mainly want to identify *activities* (as opposed to species, a well-understood problem [8]), there exists no off-the-shelf, pretrained model that can be used to extract meaningful data from this dataset. To develop a domain-specific model, scientists first need to familiarize themselves with their data and develop a vocabulary of activities within.

Existing tools do not adequately aid users in *early data exploration*—especially users who are not experts in ML—despite this being a critically-important piece of end-to-end data management. Existing video browsing systems [22, 39] focus on *known-item search*, which presumes the user already knows what they are looking for and do not support building a domain-specific model. Lancet [52] proposes to support users in building models over unstructured data by combining active learning with embedding training. However, their technique requires knowledge of ML tuning to achieve good performance on an arbitrary dataset, and it requires repeated, expensive processing over the dataset as the embedding is updated.

In this paper, we present the design, implementation, and evaluation of VOCALExplore, a system that fills this gap and supports users with early video data exploration, labeling, and model building. It is a part of our larger VOCAL system [13]. In our example, the user only needs to point VOCALExplore at their data and they can *immediately* begin exploration. Immediate interactivity is a key goal of VOCALExplore. The user only invests more effort as they see results, which is important for new domains when the user may be uncertain about the labels they wish to use and whether a good model is even possible for their data and desired labels. A key contribution of VOCALExplore is to support such initial exploration with a "pay-as-you-go" design, which avoids expensive preprocessing phases. Instead, VOCALExplore processes data incrementally as the user explores it and provides increasingly accurate results as users put more effort towards exploration and labeling. VOCALExplore enables an iterative workflow: At each step, the user either specifies which video segments they want to view or lets the system select video segments. As they watch videos, they can choose to annotate them with new or existing labels. When VOCALExplore chooses video segments for the user to view and label, it samples them in a way that yields good model quality, while avoiding extra costs

Proceedings of the VLDB Endowment, Vol. 16, No. 13 ISSN 2150-8097.
doi:10.14778/3625054.3625057

when not necessary. VOCALExplore also decides which feature representation to use for the given data. Finally, VOCALExplore does the above while hiding all significant sources of user-visible latency, providing fast response times to data exploration requests.

There are several challenges in designing a system like VOCALExplore. First, VOCALExplore brings together techniques from across the ML community that are required to support end-to-end video data exploration and model building—from video sampling to feature extraction to building models on video data. VOCALExplore combines these into a system wrapped behind a data exploration interface, which does not require any ML knowledge or tuning from the user, nor any expensive preprocessing steps.

Second, we design VOCALExplore as a pay-as-you-go system: it receives user input incrementally and must produce results incrementally as well, all without long preprocessing phases, to support low-latency data exploration. At each iteration, VOCALExplore must decide which video segments the user should label next and how to train the best model on top of these labels. While the ML community has proposed many active learning acquisition functions [42], there is evidence that no one technique is the best, and they often perform no better than random sampling as shown in [25] as well as in our evaluation (Figure 4). Further, active learning functions are more expensive than random sampling because they require preprocessing the dataset. To address this challenge, our system dynamically selects either random sampling or an active learning function based on observed class imbalance in the dataset. VOCALExplore always starts with random sampling because it is expected to perform well over uniform datasets, and it requires no preprocessing. It then switches to more expensive active learning if it observes sufficient skew in the labels. When it switches to active learning, VOCALExplore incrementally processes videos to build a candidate set over which the active learning algorithm can execute, again avoiding an expensive preprocessing step.

While it is common today to train video models using pretrained models as feature extractors, there is a lack of research exploring how to choose the best one for a new dataset. Therefore, before we begin to train a domain-specific model using the user provided labels, we are faced with the technical challenge of deciding which pretrained feature extractor to use. We show that the accuracy of domain-specific models depends on the chosen feature extractor. To address this challenge, VOCALExplore starts with a set of candidate pretrained models to be used as feature extractors. It frames feature selection as a rising bandit [28] problem to dynamically converge on the best features for a given dataset during early labeling iterations—again avoiding a separate feature selection phase—and instead integrating feature selection into the data exploration process. Note that we use the term “feature selection” to denote picking a feature extractor, rather than selecting a subset of a feature vector.

The third challenge is supporting the described functionality with low user-visible latency to make the system interactive. VOCALExplore relies on many tasks that have non-trivial latency (e.g., extracting feature embeddings from encoded videos), and naive strategies to minimize latency risk hurting model performance (e.g., eliminating model training latency by making predictions using a model trained many iterations ago). VOCALExplore addresses this challenge by using idle time to perform tasks while the user is occupied labeling videos. While the idea of leveraging background processing is not new, the key contribution of this paper lies in identifying *which* tasks to execute in the background and *when* to launch them in order to achieve a model quality that is as similar as possible to a serial execution of all tasks, all while maximally reducing user-visible latency.

In summary, VOCALExplore makes the following contributions: We design a video data exploration and labeling system that brings together state-of-the-art ML methods and wraps them with a simple data exploration interface that does not require any ML knowledge from users (Section 2). We develop an Active Learning Manager (ALM) that produces high-quality models by dynamically selecting the appropriate acquisition function and best feature extractor for each dataset (Section 3). We develop a Task Scheduler to ensure VOCALExplore produces high-quality models without significant user-visible latency (Section 4).

We evaluate VOCALExplore on standard and domain-specific datasets (Table 1). Our experiments show that VOCALExplore can achieve model performance that matches the best combination of acquisition function and feature with no preprocessing, and a user-visible latency of less than one second per labeling iteration. VOCALExplore does this while automatically deciding what features to use and how to sample video segments to be labeled.

2 SYSTEM OVERVIEW

In this section, we present the API of VOCALExplore, user workflow, and overall system architecture.

2.1 Motivating example

We first motivate VOCALExplore by describing the use case of the ecologists we partnered with who study the behavior of deer in the wild [15]. The scientists seek to understand how much time the deer spend on different activities (e.g., eating or traveling). To study these questions, the ecologists attached GoPro-style cameras to collars on the deer. These cameras collected video data for two weeks before the collars automatically fell off the deer.

Once the ecologists collected the cameras, they had access to a large quantity of video data (1.4 TB across 800k video files) that, were it labeled, would enable them to analyze their research questions. An ideal solution for these scientists would be to automatically label the videos using a machine learning model. However, no pretrained model exists for this domain-specific task. Therefore, the ecologists manually labeled a sample of the videos by temporally sampling video clips from the morning, midday, and evening, and performed their analysis on top of these labeled samples [15].

This manual labeling process is tedious, and analyzing the labeled samples is limiting, especially when the fraction of labeled data is small. As an alternative, the scientists could have manually trained a domain-specific model. This is, however, challenging for the reasons already enumerated, and because the scientists are not experts in ML. Next, we describe how VOCALExplore supports scientists to easily train a model over their videos.

2.2 API and user workflow

Workflow. Here we describe the high-level workflow users follow when using VOCALExplore. Users load their video data by specifying a set of video paths. Users can immediately start exploring and labeling their data because VOCALExplore performs no preprocessing. During this exploration, VOCALExplore samples video segments for the user to label. Initially, it randomly returns videos for the user to explore. Once the user has provided some initial labels (in the prototype, ≥5 labels), VOCALExplore additionally returns the predicted labels for each produced video segment. At any time, the user can view any subset of the video data together with VOCALExplore's predictions for those videos. The user can provide corrected labels for any errors they notice.

API. The API of VOCALExplore is shown at the top of Figure 1. **Watch** enables a user to view a video stream within a specified time window. VOCALExplore returns a sequence of consecutive video segments labeled with the activities that the system detects. Initially, labels are null. **Explore** enables system-directed exploration to efficiently build a high-quality domain-specific model. VOCALExplore returns videos (along with their predictions) that when labeled will most improve model performance. Explore optionally takes a specific label that causes VOCALExplore to return videos that will most improve its predictions for the specified class. When a user views video segments they can add labels using the **AddLabel** method.

2.3 System architecture

To enable the above workflow, VOCALExplore must support the following functionalities: sample selection to produce video segments needing labels; model training and inference to produce predictions for unlabeled videos; and feature extraction to produce inputs to models. Figure 1 shows the architecture of VOCALExplore that supports these functionalities. The *Active Learning Manager* (ALM) performs sample selection, the *Model Manager* performs model training and inference, and the *Feature Manager* performs feature extraction. Additionally, VOCALExplore includes a *Storage Manager* to manage metadata and intermediate results, and a *Task Scheduler* (TS) to coordinate these components.

The ALM (Section 3) and TS (Section 4) are the the core contributions of this paper. We defer a detailed discussion on them to the following sections; here we outline how the components interact.

Storage Manager (SM). The SM stores and retrieves all persisted data, which includes video metadata (e.g., path, duration, start time), labels, features, and models. The SM uses off-the-shelf components.

Feature Manager (FM). The FM returns feature representations of video segments. These features are used by the ALM for sample selection as well as by the Model Manager for training and inference. Features are represented by d-dimensional vectors in $\mathbb{R}^d$, and each is associated with some time period $(start, end)$ within a video.

Model Manager (MM). The MM trains models using the user-specified labels and performs inference on these models to return predictions. Given a video (`vid`), and time-interval $[t_1, t_2]$, the MM outputs a probability distribution across possible labels for that video segment. Our prototype MM maintains one model per feature extractor. The MM trains a new model whenever requested to do so by the ALM and is non-blocking: while a new model is training, the MM serves requests for labels using the previously trained model.

ALM. For each call to Explore, the ALM picks B video segments, each of duration t, that the user should label next. The ALM invokes the MM to provide predictions for the video segments being returned. We further describe the ALM in Section 3.

TS. The TS coordinates the activities of the various managers to ensure low-latency responses to user-initiated API calls while maintaining high prediction quality. We describe the associated challenges and how the TS addresses them in Section 4.

3 ACTIVE LEARNING MANAGER

The Active Learning Manager (ALM) is a central component of our system responsible for selecting the video segments that the user should label. Recall that our system focuses on tasks where a user wishes to label a small number of video segments to build a model that can serve to label the rest of the video. The ALM must address several challenges. Most importantly, our system's goal is to provide pay-as-you-go results: i.e., for each new batch of user labels, the ALM strives to maximize model quality given the labels collected so far. The ALM cannot rely on a long preprocessing phase to accumulate a large number of labels or optimally select features for a new domain. Instead, the ALM generalizes the problem of active learning to not just choosing which video segments to label (and what method to use to perform that selection), but also *simultaneously* choosing which features to use for a new domain.

The first subproblem of selecting segments to label is an active learning problem. There are many proposed acquisition functions in the active learning literature (e.g., [42]). Our goal is not to design a new active learning algorithm, but to determine when the extra cost is worthwhile in a data exploration system. Because random sampling can achieve the best model quality in some settings [25] and is less expensive, the first challenge the ALM addresses is distinguishing between when random sampling is sufficient and when an active learning acquisition function should be used for a given dataset. The key idea behind our approach is for the ALM to start with Random, observe the label distribution, and dynamically switch to other acquisition functions if the evidence suggests that active learning will outperform Random. Section 3.1 describes how the ALM chooses between these functions.

The second subproblem is feature selection. Video models use pretrained features as a starting point for new tasks. However, choosing the appropriate pretrained model from which to extract features is an open research question. We propose to dynamically select features to use for a given dataset. The key idea of our approach is to use a rising bandit method to comparatively evaluate feature quality during active learning as we describe further in Section 3.2. In contrast with feature engineering approaches [26, 46, 50], our problem is to produce a useful feature representation for the unstructured video data in the user's new domain rather than manipulating features to improve model performance.

Finally, the ALM solves both subproblems simultaneously. At each step, it makes the best decision for each independently. However, the samples selected by the acquisition function affect model performance (and therefore feature selection), and features affect the performance of active learning sampling. The ALM handles this interference by using decision methods that are tolerant to noise.

3.1 Acquisition function selection

We first discuss how the ALM solves the problem of acquisition function selection, where the acquisition function determines which video segments are selected to be labeled at any given iteration.

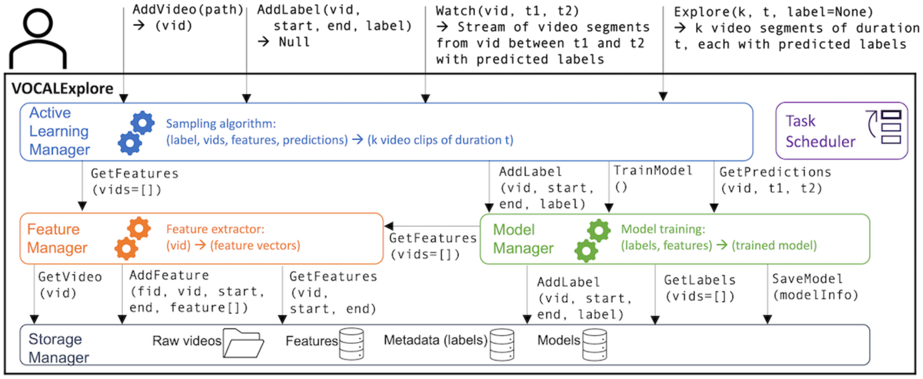

AddVideo(path)
→ (vid)
AddLabel(vid, start, end, label)
→ Null
Watch(vid, t1, t2)
→ Stream of video segments from vid between t1 and t2 with predicted labels
Explore(k, t, label=None)
→ k video segments of duration t, each with predicted labels
VOCALExplore
Active Learning Manager
Sampling algorithm:
(label, vids, features, predictions) → (k video clips of duration t)
Task Scheduler
GetFeatures (vids=[])
AddLabel (vid, start, end, label)
TrainModel ()
GetPredictions (vid, t1, t2)
Feature Manager
Feature extractor:
(vid) → (feature vectors)
GetFeatures (vids=[])
Model Manager
Model training:
(labels, features) → (trained model)
GetVideo (vid)
AddFeature (fid, vid, start, end, feature[])
GetFeatures (vid, start, end)
AddLabel (vid, start, end, label)
GetLabels (vids=[])
SaveModel (modelInfo)
Storage Manager
Raw videos
Features
Metadata (labels)
Models

Figure 1: VOCALExplore architecture. The Task Scheduler coordinates the activities of the various managers.

Problem. The ALM is given a set of video segments, $v \in V$. The video segments depict various activities $a \in A$, and these activities may be skewed, meaning that some appear more frequently than others. It is possible for a single video segment to contain multiple activities, or no activities. We are also given a labeling budget, B, which designates the number of video segments a user is willing to label. This budget is incremental and is not fixed. For example, a user may initially set B=20 and then give VOCALExplore another B=10 if they are willing to label more. The ALM uses the labels to train a model M, which in the prototype is a linear model.

The ALM balances the two goals of maximizing model quality, (G1), and producing pay-as-you-go results, (G2). For G1, we consider the average model quality across all classes the user has applied to video segments. The prototype maximizes the macro F1 score of the model, though other metrics could be used. For G2, the ALM strives for interactivity and low latency in response to API calls by avoiding expensive preprocessing steps that block user interactions.

3.1.1 *Baselines.*

We consider the use of individual acquisition functions as baselines. The relative performance of any acquisition function depends on the dataset, but the cost of each function is partially determined by the inputs the function requires (the other component of cost is the processing done on top of the inputs).

The most naive strategy is Random, which randomly selects the B video segments. This is cheap because its inputs are video metadata (e.g., duration) rather than features extracted from the video frames. However, if the activities in the dataset are highly skewed, then random sampling will not find many examples from activities that rarely occur, which hurts G1 because the model will perform poorly on these rare classes. Additionally, as we observe experimentally, random sampling over skewed data causes the user to label large amounts of the same activity type and very few rare activities. We posit that having the user label more diverse activity types is more in line with supporting users in early data exploration.

More sophisticated baselines use active learning techniques that take as inputs features, and possibly model outputs. These strategies require an expensive, one-time preprocessing step to extract features from all of the video segments V. Uncertainty-based techniques additionally require performing inference over all $v \in V$. This preprocessing hurts G2 because it results in a large amount of initial latency, even if the user only makes a small number of API calls, and the feature extraction and inference tasks over all of the videos result in high latency for API calls. However, active learning acquisition functions can improve model performance over naive random sampling [41], especially for skewed datasets where random sampling will have low label diversity.

3.1.2 *Our approach (VE-sample).*

The ALM resolves these tradeoffs by casting acquisition function selection as a binary decision between Random or an active learning-based acquisition function. It dynamically switches to a more expensive active learning function only when it is expected to improve model performance and label diversity. The ALM strategy, which we call VE-sample, initially uses random sampling to select the B video segments to be labeled because it is fast and requires no preprocessing (G2), and it performs well for uniform datasets (G1). VE-sample dynamically switches to active learning if it observes skew in the labels it collects. This results in better label diversity for the user and, more importantly, improves model performance on rare classes (G1).

When our prototype switches to active learning it uses cluster-margin sampling [12] which combines uncertainty and diversity sampling. Our prototype also implements the greedy coresets algorithm [41] (Coreset), which is a density-based acquisition function that has been shown to work well in a batch-labeling setting and is designed to find diverse examples. By default the ALM uses Cluster-Margin sampling for active learning because in our experiments it always performs at least as well as Coreset.

To decide whether the labels are sufficiently skewed to switch to active learning, VE-sample uses the k-sample Anderson-Darling test [40] which is a statistical test for comparing discrete distributions. VE-sample compares the label distribution observed so far to a baseline uniform distribution and switches to active learning when $p \leq 0.001$. We use this small p-value because the label distribution is initially noisy when there are a small number of labels. We want to switch away from random sampling only when we are highly confident that the distribution is in fact skewed.

Other statistical tests are possible. For example, we could also say that a dataset is skewed if the imbalance ratio [34] (i.e., the ratio between the frequency of the majority and minority classes) is large. If there are k classes, and the multinomial distribution has parameters $p \in \Delta_k = \{p \in \mathbb{R}_+^k : \sum_{i=1}^{k} p_i = 1\}$, we can say a distribution p is skewed if $\min_i p_i < \frac{1}{mk}$ for some multiplicative threshold m. m is a lower bound on the imbalance ratio because the majority class must have frequency $\geq 1/k$. For this frequency-based approach we set the p-value to be equal to an upper bound on the probability of incorrectly classifying a dataset as skewed. Details of how this bound is derived are described in our extended technical report [14]. The benefit of using the frequency test is that its p-value will not grow smaller solely based on an increasing number of data points if the dataset is not perfectly balanced. Whereas the Anderson-Darling test will return a small p-value for slight class imbalances (e.g., 51% class A and 49% class B) given sufficient labels, the frequency test with high probability will not detect this as skewed even in the limit of infinite labels. We show in Section 5.2 that this frequency-based test matches the F1 scores achieved when we use the Anderson-Darling test.

Interestingly, we empirically find that the VE-sample approach has the additional effect of producing a more diverse set of video segments for the user to label, compared with using random sampling alone. A diverse labeled set benefits model performance, but it also makes the labeling task more interesting for the user. Given $n_a, a \in A$, the number of labels for each activity type, we measure label diversity as $S_{max}=\frac{\max_{a\in A} n_a}{\sum_{a\in A} n_a}$, which represents the fraction of labels that come from the most-seen activity. A lower S_{max} indicates a higher diversity of labels. Other measures are possible.

Finally, the ALM addresses G2 by incrementally processing videos. The ALM extracts features from labeled videos to train models, and from sampled videos to make predictions, so the amount of processing is proportional to the amount of user interaction. For Random, this requires only processing the videos that contain the B video segments returned from Explore. However, when VE-sample switches to active learning, the active learning algorithm requires a set of candidate features. The ALM balances active learning quality and visible latency through a hyperparameter, X. When VE-sample is using active learning and the user requests B video segments from Explore, VE-sample ensures this set contains features from X additional videos. We evaluate the impact of the choice of X on both latency and model quality in Section 5. As described in Section 4, the Task Scheduler hides the latency of this incremental processing so it does not affect interactivity.

3.2 Feature extractor selection

As discussed, VOCALExplore uses pretrained image and video models as feature extractors because they have a favorable cost/quality tradeoff (model inference is highly optimized on GPUs), and pretrained models are commonly used to initialize domain-specific models [18]. The MM trains one model per candidate feature.

3.2.1 *Problem.* We observe that the performance of feature extractors varies depending on the dataset and task. As shown in Figure 5, some feature extractors perform much better than others on a given dataset, and the best feature varies across datasets.

The ALM is responsible for finding a feature extractor that leads to high-quality models when trained over the user-provided labels. Each feature extractor takes as input one or more frames and outputs a feature vector. The problem that the ALM solves is as follows: when starting with pixel data and multiple candidate feature extractors, how should it pick the extractor whose outputs lead to the highest model quality for the given domain. This is in contrast to traditional feature engineering techniques [26, 46, 50] that take as input features applicable to the domain and attempt improve model quality by transforming these input features. By default, VOCALExplore uses a pool of video and image pretrained models as candidate feature extractors (Table 2) with differing architectures and pretraining, making some models better suited for adapting to a new domain than others. For example, CLIP is an image classification model, and we observe its features perform better on datasets where activities can be determined by looking at individual frames (e.g., K20 in Table 1). To extract features for a particular video, the FM performs inference on sampled clips or frames (for video or image models, respectively) to extract feature vectors. Each feature vector is associated with a feature ID, video ID, and some time span corresponding to the input frame(s): $(fid, vid, start, end, vector)$.

The ALM must pick a feature to use at each step when it returns predictions for video clips because the feature determines which model is used to make predictions. The ALM must also pick a feature to use if VE-sample uses active learning (see Section 3.1). Picking a feature that performs poorly leads to incorrect predictions, and, in the case of active learning, suboptimal clip sampling. Therefore, the ALM dynamically selects the features to use based on the empirical performance on each dataset.

3.2.2 *Naive strategies.* A first naive strategy is to concatenate all of the possible features into a single feature vector. However, this requires a large amount of compute resources to extract all features from all videos (as shown by the latency of the VE-lazy (PP) strategy in Figure 8). Further, we do not observe an improvement in performance over the best single feature, as shown in Figure 5.

A second naive strategy would be at each step to use the feature that is performing best. At each step, the ALM could extract *all* possible features from all labeled video clips, and then train a different model for each feature. It would pick the feature that empirically performs best. This second strategy is also inefficient because it requires extraction, training, and evaluation for every feature.

3.2.3 *Bandit strategies.* While the ALM initially must explore all possible feature extractors as in the second naive strategy, we want to quickly converge on one of the best ones. Once a feature is picked, all compute resources can be dedicated to extracting just that feature from the remaining video segments, and models are only trained using that feature. The problem then becomes how to converge on one of the best features. On the surface, this appears to be a problem that can be solved by Multi-Armed Bandit (MAB) approaches: each feature extractor is an arm, model performance is the reward, and we want to exploit the feature extractors that lead to the best model performance. However, MAB techniques assume stationary reward distributions (i.e., the reward for pulling an arm is independent of the number of times that arm is pulled). This is not true for our use case because model performance is expected to improve as the amount of training data increases. If a feature performs poorly in early rounds, we do not want to eliminate it solely based on early performance values because it is possible that it will improve once there are more labels.

Our setting is that of *Rising Bandits* [28]. Rising Bandits do not assume that the rewards for each arm are stationary; rather, they are assumed to be increasing in a concave manner as the arm is pulled. Under these assumptions, the expected performance of each arm after some number of examples can be bounded, and arms can be eliminated when the upper bound on their expected reward is lower than the lower bound for some other arm.

The original Rising Bandit algorithm [28] that the ALM adapts works as follows: The algorithm proceeds in a series of rounds. At each step, it computes the current model quality for each candidate feature. Then, it computes lower (l_f) and upper (u_f) bounds for the expected performance after T timesteps. The lower bound is taken to be the current value because we assume the quality increases over time. The upper bound ($u_f = l_f + \omega_f \times (T - t)$) is taken to be the lower bound plus some delta computed as slope (ω_f) multiplied by the number of remaining timesteps ($T - t$), which is a linearization at the current time t step evaluated at T. Because of the concavity assumption, the linearization is an upper bound on the true reward. Finally, features are eliminated when their upper bounds are below the lower bound of any other feature. Note that the algorithm from [28] was proposed in a different setting from ours and thus the guarantees do not directly transfer. In particular, the "reward" in our setting is the performance of the chosen arm with T points, while the "reward" in [28] is the performance of the chosen arm with however many points were allocated to that arm.

3.2.4 *VOCALExplore adaptations to Rising Bandits.*

The ALM must resolve three challenges before applying the Rising Bandit framework. First, measured model performance is noisy. While it is expected to increase on average over time, individual time steps may have a decrease in performance if the added labels temporarily make it more challenging for the model to distinguish classes. Second, measured model performance is not guaranteed to increase in a concave manner because the training set grows over time and because the ALM may switch to active learning from random sampling. Finally, the user does not initially have a labeled validation set, but the ALM still must reliably estimate model performance.

To resolve the first challenge of noisy performance data, the ALM performs smoothing on top of the measured values. The goal is to capture the trends in performance but avoid any temporary spikes or dips. The prototype uses exponential weighted moving average (EWMA) smoothing, but other techniques are possible. The prototype also waits 10 iterations before beginning feature selection because model performance is particularly noisy in early iterations when there are a small number of labels.

To resolve the second challenge of non-concave performance increases, the ALM uses the proposed solution from the Rising Bandits algorithm [28]. Recall that the algorithm computes the upper bound using the slope to estimate the value after some number of steps into the future. Rather than computing the upper bound using a slope over the current and immediately previous timesteps t and $t{-}1$, the ALM computes a smooth growth rate over a larger window of size C: t and $t{-}C$.

To resolve the final challenge of the lack of a validation set, the ALM estimates the performance of features using cross-validation. The ALM creates three train/test splits over the labels it has collected so far and averages the performance across these splits. While training and evaluating multiple models is more expensive than evaluating a single model over a held out validation set, the ALM only does this at the start of exploration when there are a small number of labels until it picks the best feature (which usually requires fewer than 150 labels in our experiments). Training linear models with a small number of examples is fast, so the additional overhead is limited. The prototype only evaluates k-fold validation over classes with at least three labeled instances to ensure each class is present in each training and test split.

While the original algorithm in [28] evaluates one arm at each time step, our modified algorithm evaluates all candidate features at each time step because the new labels provided by the user can be used to update the model for all features.

3.2.5 *Hyperparameter setting.*

The hyperparameters the ALM uses for feature selection are: C (slope smoothing window), T (timestep used to compute the upper bound), and w (smoothing span for EWMA; $\alpha{=}2/(w{+}1)$). As discussed in Section 5.3, the sensitivity of C and w is low; a range of values provide similarly good performance. This agrees with the findings of [28] that the performance of their algorithm is not sensitive to C. Therefore, the ALM uses a "moderate" amount of smoothing and sets $w{=}5$ and $C{=}5$.

T is the time point at which the upper bound is computed. Larger T values lead to higher upper bound estimates, therefore features are eliminated more slowly. Using a larger T value is more robust against non-concave performance curves because when the slope is small at early steps, the upper bound will still be high enough to not eliminate the feature before its slope later increases. However, larger T values require more compute power because a larger number of features will be extracted and evaluated for more steps. Therefore, our approach is to set T to a small value (e.g., $T{\leq}50$) in resource-constrained settings. This may not lead to selection of the optimal feature, but our evaluation shows that one of the best features is still selected with high probability. In settings where resources are not constrained, T can be set to a larger value (e.g., $T{=}100$) because there are sufficient resources to evaluate more features for additional steps, and therefore allow the ALM more time to attempt find the single best feature (though, using a larger T doesn't *guarantee* finding the best feature).

4 TASK SCHEDULER

The Task Scheduler is a priority scheduler that runs in the background and schedules VOCALExplore's tasks on the available compute resources. We consider a setting where there are limited resources, so only a subset of submitted tasks can execute at once. From Section 3.1, goal G2 states that VOCALExplore should ensure interactivity and low latency in response to API calls. VOCALExplore is intended to support data exploration, so it needs to minimize any user-perceived latency because increased latency is known to decrease user interaction [30]. Naive and lazy scheduling of VOCALExplore's tasks results in substantial latency as we discuss in this section (and show in Section 5). The goal of the Task Scheduler is to optimize that latency without compromising the model quality seen by the user whenever they make API calls.

The Task Scheduler achieves this by making non-critical tasks asynchronous and performing just-in-time model training (Section 4.1), and by eagerly performing feature extraction while the user is occupied labeling (Section 4.2). These optimizations systematically target the principal sources of user-perceived latency.

Background. VOCALExplore has five types of tasks: feature extraction (T_f), model training (T_m), model inference (T_i), feature

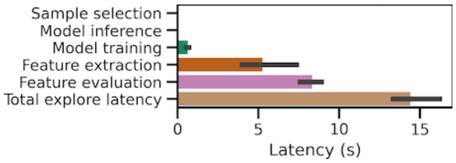


Figure 2: Median task latency for early Explore steps. The error bars show the interquartile range.

evaluation (T_e), and sample selection (T_s). Each Explore call corresponds to multiple tasks of multiple types: VOCALExplore must first select a batch of video segments for labeling (this represents one task T_s per sample); extract features from the sampled segments if not already available (one task T_f per sampled video segment); perform inference with the latest model (one task T_i per sampled video segment); collect the labels from the user; train a new model (T_m); and evaluate feature quality for remaining features (one task T_e per feature; see Section 3.2). Additionally, if VOCALExplore needs to sample video segments using active learning instead of random sampling, it needs to sample more video segments than the user-requested number and extract features from the extra samples before selecting segments to return to the user for labeling, requiring a larger number of T_f tasks. T_f and T_e are the most expensive tasks as shown in Figure 2, which illustrates the latency each task contributes to a serial schedule when using random sampling. We performed a sequence of Explore steps with B=5 and t=1 on the Deer dataset (Table 1) and measured latency across steps 5-10.

Baseline. Let T_{serial} be the API latency of a call to Explore with a serial schedule, k the number of features still under consideration, and B the number of video segments labeled each iteration. With some abuse of notation, let's consider each T_x to represent not just the type of task but also the time to execute one such task. We then have: $T^{random}_{serial} = B(T_s + T_f + T_i) + T_m + kT_e$ for random sampling and $T^{active}_{serial} = (B + X)T_f + B(T_s + T_i) + T_m + kT_e$ when using active learning, where X is the number of extra samples that the ALM uses for active learning (Section 3.1). There are still only B T_s tasks because we only must select B samples (e.g., in Coreset, we perform B max-distance calculations).

The Task Scheduler does not minimize T_{serial} directly. Rather, we observe that the user spends a non-negligible amount of time, BT_{user}, labeling video segments after each call to Explore. The Task Scheduler exploits that time to do useful work.

Problem statement. Let T_{total} be the time needed for VOCALExplore to return video segments to a user in response to a call to Explore plus the time for the user to label the returned B video segments, so it represents the total time elapsed during a labeling session. The user returns labels $L_1, \ldots, L_B$. Given a sequence of calls to Explore, the goal of the Task Scheduler is to minimize, at each iteration u, the user-perceived latency defined as: $T^u_{visible} = T^u_{total} - BT_{user}$, subject to maintaining good model quality. For the latter, given Q^u_{serial} the model quality (measured by any metric; we use macro F1 score) seen by the user for a serial schedule at iteration u, and $Q^u_{optimized}$ the model quality with the optimized schedule at the same iteration u, the Task Schedule seeks to ensure that $Q^u_{serial} - Q^u_{optimized} < \epsilon$. In our system, we do not start with a fixed ϵ but rather develop task scheduling approaches that empirically yield a small ϵ value.

4.1 VE-partial strategy

Our first step towards an optimized strategy, VE-partial, uses the insight that not all tasks are equally critical for providing a response to API calls. Only selecting video segments, T_s, extracting features from them if not already available, T_f, and performing model inference, T_i, are required to return from Explore. VOCALExplore hides model training latency by performing inference over the most recent model that has already been trained. Similarly, feature evaluation tasks do not block Explore; VOCALExplore updates the set of candidate features in the background as T_e tasks complete. The VE-partial strategy makes model training (T_m) and feature evaluation (T_e) asynchronous tasks, which reduces the user-perceived API latency to $T^{random}_{VE-partial} = B(T_s + T_f + T_i)$, and $T^{active}_{VE-partial} = (B + X)T_f + B(T_s + T_i)$. The quality of predictions is $Q^{u-\delta}_{VE-partial}$, where δ indicates how stale the model is.

The challenge the Task Scheduler addresses is to ensure that $Q^{u-\delta}_{VE-partial}$ is close to the quality achieved with the serial schedule. Using a model trained many iterations ago ($\delta \gg 0$) will not suffice because its quality is too low. Scheduling a new model training task after each new label is also not desirable. While this approach ensures that $\delta \approx 0$, it results in a factor of B more model training tasks, which causes congestion in the task queue. This approach also wastes resources because many models will never be used; when $T_m < (B-1)T_{user}$, multiple model training tasks will be queued and finish during a single iteration, but the ALM will make predictions using just the latest one.

The Task Scheduler addresses this challenge using "just-in-time" model training to minimize δ while still avoiding user-visible latency due to model training. The ALM tracks user labeling time (T_{user}) and model training latency (T_m). The ALM schedules a model training task after receiving $\max(0, B - \lceil T_m/T_{user} \rceil)$ labels because this ensures the model will be ready for inference by iteration u+1. When $T_m < T_{user}$, the ALM schedules a training task while the user labels the last example (i.e., after receiving L_{B-1}), so it makes predictions using a model trained with all but one label. If model training takes longer than an entire exploration iteration ($B - \lceil T_m/T_{user} \rceil < 0$), then the ALM schedules a model training task while the user labels the first sample. This model will not be ready for inference by u+1, but it will be ready by $u + \lceil T_m/(BT_{user}) \rceil$.

The VE-partial strategy reduces latency by making low-priority tasks asynchronous, and it maximizes model quality by scheduling "just in time" model training based on observed latencies.

4.2 VE-full strategy

The VE-partial strategy still has non-negligible latency due to feature extraction. $T_f \gg T_i$ because feature extraction operates over encoded videos, which requires expensive preprocessing, while inference operates over already-extracted feature vectors.

This leads to the Task Scheduler's second optimization: eager feature extraction. Strategy VE-full eagerly schedules feature extraction tasks (T_{f^-}) for unlabeled videos whenever the task queue is empty. These tasks have the lowest priority, so if any other task is scheduled while T_{f^-} is still queued, it will execute first. T_{f^-} tasks perform the same work as T_f, just at a lower priority. Initially,

Table 1: Datasets

Dataset	# classes	Skew	Train videos	Eval videos
Deer	9	Skewed	896	225
K20	20	Uniform	13326	976
K20 (skew)	20	Skewed	1050	976
Charades	33	Skewed	7985	1863
Bears	2	Uniform	2410	722
BDD	6	Skewed	800	200

Table 2: Features used by VOCALExplore.

Feature	Type	Architecture	Pretrained
R3D [45]	Video	Conv. net	Kinetics400
MViT [17]	Video	Transformer	Kinetics400
CLIP [37]	Image	Transformer	Internet images
CLIP (Pooled) [37]	Image	Transformer	Internet images
Random	Video	Transformer	None

there are no unlabeled video segments from V with features extracted: $S{=}\emptyset$. The ALM randomly samples a set s of unlabeled video segments and schedules feature extraction tasks for all current candidate features, which results in a total of $k{\cdot}s$ T_{f}- tasks. When these tasks complete, $S \leftarrow S \cup s$. The prototype sets $|s|{=}10$ to amortize the cost of setting up a feature extraction pipeline across multiple video segments while still completing the task within a few seconds.

The VE-full strategy has user-visible latency $T_{VE-full} = B(T_s + T_i)$ for both random sampling and active learning because the ALM uses S for both to eliminate feature extraction latency T_f.

Quickly converging to a single feature (Section 3.2) enables the most efficient growth of S because the number of T_{f}- tasks is proportional to the number of candidate features. Growing S is desirable because it enables better active learning performance (shown in Section 5) and reduces prediction latency. It still is in the spirit of "pay-as-you-go" because the extra processing only happens while the user is interacting with the system.

5 EVALUATION

We perform an evaluation of VOCALExplore. First, we show that compared to baselines, VOCALExplore achieves a high F1 score with the lowest latency, even while automatically performing feature and acquisition function selection (Section 5.1). Second, we demonstrate the effectiveness of the ALM's acquisition function selection process (Section 5.2). Then we demonstrate that the ALM's feature selection algorithm picks one of the best features within a small number of steps (Section 5.3). Finally, we evaluate the effectiveness of the Task Scheduler to show that it ensures low latency without hurting the F1 score (Section 5.4).

Implementation details. The prototype is built using Python 3.8.10. The storage manager stores video metadata, labels, and model metadata in DuckDB 0.5.1 [36]. It stores feature vectors in Parquet files, and it uses PyTorch [35] to train models. It uses the filesystem to store and retrieve encoded video files. Videos are stored on hard drives, while all other data is stored on the local SSD. The feature manager uses NVIDIA DALI [5] to accelerate feature extraction when a GPU is available, otherwise it uses PyTorchVideo [16]. In the evaluation we perform feature extraction on the GPU, and the model manager trains linear models.

Evaluation setup. We conduct all experiments on a compute cluster. When measuring runtimes, we request one node with eight Intel Xeon Gold 6230R CPUs @ 2.10GHz, 61GB of RAM, and one NVIDIA A40 GPU. This setup was chosen to approximate the memory, CPU, and GPU setup of a "p3.2xlarge" EC2 instance on AWS.

Datasets. We evaluate VOCALExplore on the datasets shown in Table 1. First, we evaluate on the Deer dataset which contains 10-second video clips captured from a camera attached to a collar on a deer [15]. We use a subset of the full dataset that we manually labeled, which covers one day for a single deer. These clips show six activities that occasionally co-occur: bedded, chewing, foraging, grooming, looking around, and traveling. The activities are highly skewed towards the "bedded" activity. We create 5 train/eval splits by ordering the video clips temporally and taking every fifth one to be in the test set. Results are averaged across these splits.

We also evaluate on subsets of Kinetics700 [44], which is a standard video dataset comprising 700 human action classes. K20 contains 10-second video clips showing activities from 20 classes taken from the Kinetics700 dataset. We pick classes that do not appear in Kinetics400 to avoid overestimating performance for features that are extracted from models pretrained on Kinetics400. K20 is not skewed, however we introduce skew to create K20 (skew). The classes in the skewed dataset follow a Zipfian distribution with s=2. The most common activity has 650 videos and the least common activity has 3 videos. We create 10 training instances of K20 (skew) by permuting the classes. Results are averaged across these 10 instances. We use videos from the Kinetics validation set for evaluation, which is not skewed (even for K20 (skew)).

Charades [43] consists of 30-second videos showing 157 distinct activities. For our experiments, we simplify the task to identifying which of the 33 verb categories appear in each video.

The Bears dataset consists of 5-second video clips captured from 19 camera traps in Alaska, primarily at night. The task is to determine whether or not each video clip contains a bear.

Finally, the BDD dataset [51] consists of 40-second video clips captured from moving cars. We extracted object detections from 1 fps using a Faster R-CNN model [29], and the task is to determine which objects (car, truck, person, bus, bicycle, and/or motorcycle) the 1.5 seconds covered by each feature vector contains.

Feature extractors. We initialize VOCALExplore with five candidate feature extractors shown in Table 2. We pick these feature extractors to cover image- and video-based models with a variety of architectures. For all of the features with input type "video", we use a sequence length of 16 (number of frames fed into the model), a stride of 2 (gap between frames in the sequence), and a step of 32 (gap between sequences). For the CLIP feature, we sample the middle frame out of every 32 frames so the feature aligns with the middle of the video feature windows. For the CLIP (Pooled) feature, we apply the CLIP model to every other frame from a window of 32 frames and perform max-pooling over the frame-level features. All of the features have 512 dimensions, except for MViT and Random which have 768 dimensions. We include the Random feature (which uses the same architecture as MViT but with randomized weights) to show that VOCALExplore handles low-signal features correctly.

Metrics. We evaluate model performance using macro F1 score because it is a standard evaluation metric. The F1 score is computed

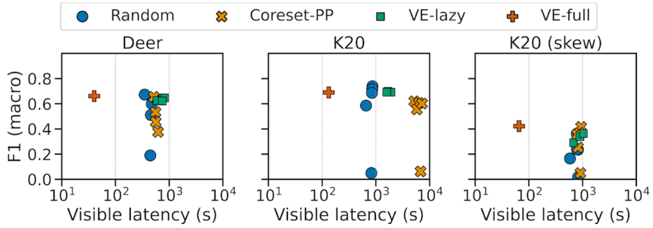


Figure 3: Average F1 and cumulative visible latency (shown with a log-scale) after 100 Explore steps. Coreset-PP includes the preprocessing time to extract each feature, and each point for Coreset-PP and Random represents a single feature. VE-full provides nearly the best model quality with the lowest visible latency.

over the held out evaluation set after training a model on the labels collected so far at each step. We initialize VOCALExplore with the entire vocabulary that exists in the evaluation set so that it trains models that predict all evaluation classes, even when some classes don't have labels yet. We evaluate latency by measuring the wall clock time taken for VOCALExplore's API calls to return.

For the experiments below, we simulate a labeling task by creating an oracle "user" that labels video segments with their ground-truth labels. Labeling proceeds in a sequence of steps where we add five 1-second labels (which corresponds to Explore(B=5, t=1)).

5.1 End-to-end performance

We first demonstrate that VOCALExplore achieves the best balance between visible latency and F1, as shown in Figure 3 (note that latency is shown with a log-scale). This experiment executes 100 calls to Explore as described above, and we measure the cumulative visible latency. Random and Coreset-PP use the serial scheduler. Random performs random sampling over the videos, and we include a point for each candidate feature. All of Random's latency comes from making predictions over the video clips returned from Explore because its sampling latency is negligible. Coreset-PP uses Coreset sampling to select videos, and we include a point for each candidate feature. The cumulative latency includes the time it takes to extract each feature from all of the videos as a preprocessing step. VE-lazy performs acquisition function and feature selection as described in Section 3, but without the scheduling optimizations described in Section 4. VE-lazy incrementally extracts features from X additional videos if needed for active learning, as described in Section 4. The graphs show a point for each of $X \in [10, 50, 100]$. VE-full includes all of the scheduling optimizations described in Section 4. This experiment simulates the user taking 10 seconds to label each video clip, which is time VE-full uses to perform feature evaluation, train models, and eagerly extract features from videos. VE-full does not specify X; when the ALM switches to active learning it uses the features that have been eagerly extracted.

VE-full's model performance matches or exceeds VE-lazy with a fraction of the visible latency, and its performance is close to the performance achieved by the best combination of acquisition function and feature. VE-full beats the model performance of VE-lazy on K20 (skew) because VE-lazy performs Coreset over a small sample of videos ($X \in [10, 50, 100]$), while VE-full extracts features from more videos in the background, and Coreset performs better over this larger sample. On the uniform K20 dataset, VE-lazy has more latency than Random because it performs feature evaluation. We discuss why the model quality of VE-full is lower than the best Random point for K20 in Section 5.3. While Coreset-PP has higher visible latency than Random, the difference is less on Deer and K20 (skew) than K20 for two main reasons. First, there are fewer total videos, so there is a smaller difference between the number of videos processed during the 100 Explore steps and the number of videos processed during preprocessing. Second, there is overhead to creating each DALI feature extraction pipeline, so preprocessing all videos at once is more efficient because it can use a single pipeline.

The optimizations from Section 4 could be applied to Random and Coreset-PP to reduce their latency, however that does not solve the problem of how to pick the correct combination of acquisition function and feature for an arbitrary dataset. As shown in Figure 3, model quality differs significantly across combinations.

5.2 Acquisition function selection

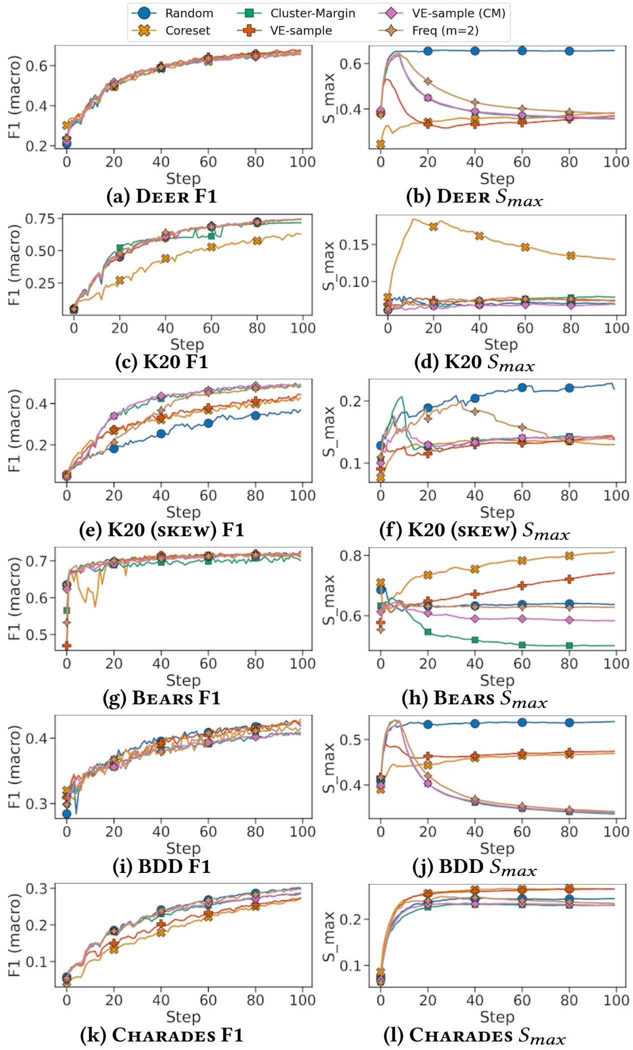
We now focus on the effectiveness of the ALM's acquisition function selection, as discussed in Section 3.1. We compare against baselines of using a fixed function: either always performing Random, Coreset [41], or Cluster-Margin [12] sampling. VE-sample picks between Random and Coreset at each iteration as described in Section 3.1, while VE-sample (CM) picks between Random and Cluster-Margin. Freq. also picks between Random and Cluster-Margin but uses the frequency-based test described in Section 3.1. For this experiment, we show results only for the best feature (Figure 5). We evaluate with R3D for Deer, MViT for K20 (skew) and Charades, and CLIP (Pooled) for K20, Bears, and BDD.

We measure performance by both the macro F1 score of the model, as well as a diversity metric S_{max}, which computes the fraction of labels that come from the single most-seen activity (see Section 3.1). A smaller S_{max} indicates that the user sees more diverse examples, which makes the labeling task more interesting.

First, Figure 4 shows that Cluster-Margin (and therefore VE-sample (CM)) always perform at least as well as Coreset and VE-sample. Therefore, we limit the rest of our discussion to Random, Cluster-Margin, and VE-sample (CM).

Looking at the uniform datasets of K20 and Bears, we observe that Random produces models with the same F1 score as Cluster-Margin. Therefore, it is unnecessary to sample these datasets with the more expensive active learning technique. Looking at the skewed datasets, we observe that using active learning boosts the F1 score above Random for K20 (skew). We also see improved (i.e., lower) S_{max} metrics for the skewed datasets when using Cluster-Margin. Therefore, it is useful to use active learning on skewed data because it is possible the model performance will be improved, and the user is likely to see a more diverse set of examples to label. We observe that VE-sample (CM) matches the performance of the best technique on each dataset by detecting whether the labels are skewed and switching to active learning if appropriate.

Finally, we observe that using the frequency-based method for determining whether a dataset is skewed leads to similar results as the Anderson-Darling k-sample test, though it is slightly more conservative and takes longer to switch to an active learning sampling method. This can be modified by adjusting m; we don't show



(a) Deer F1

(b) Deer S_{max}

(c) K20 F1

(d) K20 S_{max}

(e) K20 (skew) F1

(f) K20 (skew) S_{max}

(g) Bears F1

(h) Bears S_{max}

(i) BDD F1

(j) BDD S_{max}

(k) Charades F1

(l) Charades S_{max}

Figure 4: VOCALExplore's data sampling method yields models with the highest F1 scores and samples from a diverse set of classes (S_{max}, lower is better) across datasets.

Table 3: Feature selection correctness.

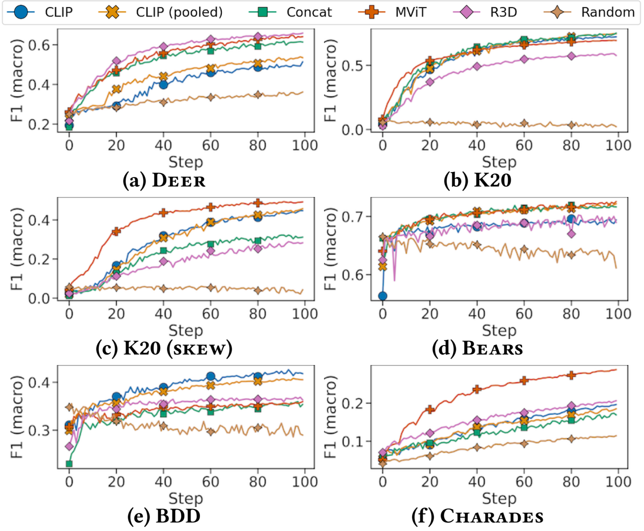
	Deer	K20	K20 (skew)	Bears	BDD	Charades
T = 20	1.00	1.00	0.98	0.97	0.50	0.87
T = 50	0.99	1.00	1.00	0.95	0.69	0.92

the results to avoid crowding the graphs, but using m=1.5 leads to curves that more closely match VE-sample (CM).

5.3 Feature selection

We now evaluate the effectiveness of the ALM's feature selection algorithm. We measure the correctness (i.e., how *frequently* do we pick one of the best features) and the efficiency of the selection (i.e., how *quickly* do we pick a feature). We initialize VOCALExplore with the five candidate feature extractors from Table 2.

We first evaluate the correctness of feature selection. To measure the quality of each feature, in Figure 5, we compute the macro F1 score for each feature across 100 labeling iterations (using VE-sample (CM) to pick video segments). It includes Concat to show that concatenating all of the potential features does not improve performance over the best single feature. Based on these results, we use the following rules when determining the correctness of feature selection. For the Deer dataset, we consider selecting either R3D or MViT to be a correct decision. For K20 and Bears, we consider any of MViT, CLIP, or CLIP (Pooled) to be a correct decision. For K20 (skew) and Charades we consider only MViT to be correct. For BDD we consider CLIP or CLIP (Pooled) to be correct. In this experiment we use C=5, w=5. We discuss sensitivity to hyperparameter values at the end of this section.



(a) Deer

(b) K20

(c) K20 (skew)

(d) Bears

(e) BDD

(f) Charades

Figure 5: Macro F1 score when using the VE-sample (CM) sampling method, which shows that the best feature varies across datasets. "Concat" refers to concatenating all of the features into a single feature vector.

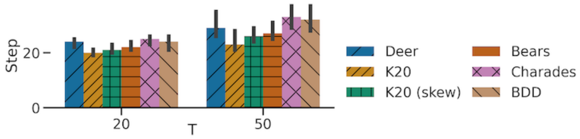
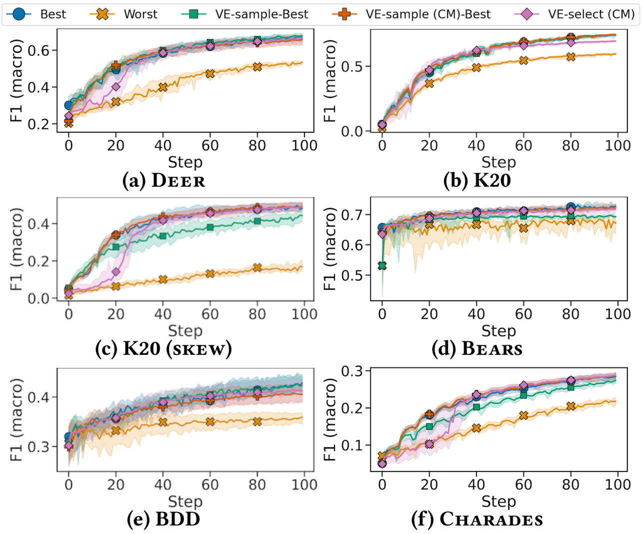


Figure 6: Median feature selection step when C=5 and w=5. Error bars show the IQR. VOCALExplore converges to a single feature within a reasonable number of steps.

Table 3 shows that the ALM picks a correct feature at least 92% of the time (excluding BDD) when the time horizon is long enough (T=50). When the algorithm picks incorrectly, it primarily picks the next-best feature (e.g., one of the CLIP features for Deer or K20 (skew)). The algorithm selects incorrect features for BDD some of the time because all features perform similarly until later iterations when CLIP and CLIP (Pooled) start to perform better. Therefore, despite the correctness measure being low, the F1 score achieved is close to the best as shown in Figure 7e. The algorithm struggles with Charades due to the noise introduced by evaluating with k-fold over the large number of classes; correctness is ≥95% when evaluating with the full test set as described at the end of this section.



(a) Deer

(b) K20

(c) K20 (skew)

(d) Bears

(e) BDD

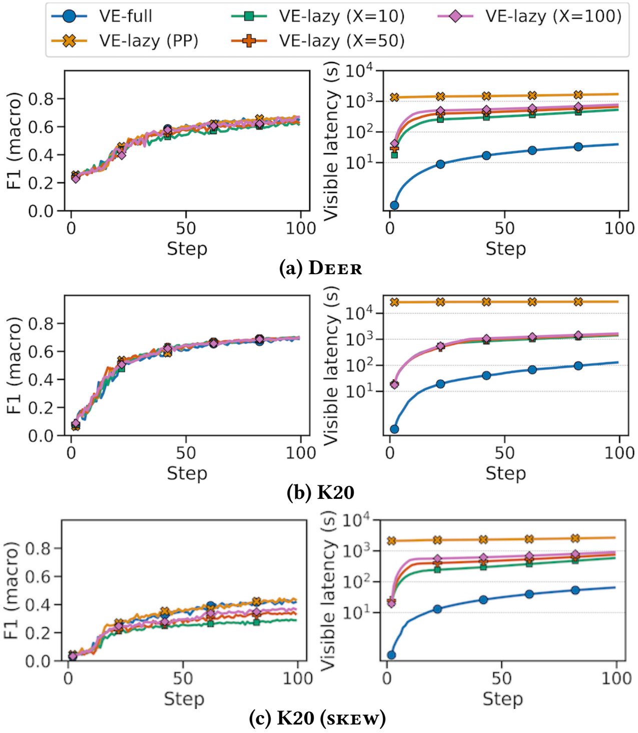
(f) Charades

Figure 7: Macro F1 score when performing feature selection compared to the empirically best- and worst-performing sampling methods and features (excluding the Random feature). We also compare against VE-sample and VE-sample (CM) sampling methods on the best feature. VOCALExplore initially has poor F1 performance as it explores suboptimal features but catches up to the best strategies within 30 steps. The shaded region shows the IQR.

The performance over Charades with k-fold can be improved to 98% correct by using stronger smoothing (w=7, C=7).

Figure 6 shows that the ALM picks a single feature within a small number of iterations. Convergence is faster when T=20 than T=50 because the upper bounds on the expected performance have lower values, so features are eliminated more quickly. Even at T=50, features are selected within about 30 steps. We use T=50 in the rest of the experiments.

We also evaluate the model quality as the ALM performs feature selection (VE-select). Figure 7 shows that while VOCALExplore initially has sub-optimal quality as it explores features, it catches up to the best-performing strategies within approximately 30 steps. We compare against Best and Worst, which correspond to the empirically best- and worst-performing combinations of sampling methods and features (excluding the Random feature) to show the range of expected values. We also compare against VE-sample and VE-sample (CM) on the best feature (VE-sample-Best and VE-sample (CM)-Best, respectively). We observe that initially VE-select's performance is close to the worst strategy because it has poor-performing features as candidates which produce models with low F1 scores. The VE-select curve exhibits an "S" shape, where once it converges to a single feature, performance catches up to the best strategies. While K20 does not converge to a single feature until 30 steps, the model quality improves before then because bad features are eliminated early. K20's final model quality is slightly lower than the best because it picks MViT 98% of the time, and MViT has the highest quality when there are few labels but not when there are a larger number of labels (as shown in Figure 5b). Because we use a small T value to encourage quick convergence to one feature, the ALM's feature selection algorithm is biased towards features that perform well in early iterations.



(a) Deer

(b) K20

(c) K20 (skew)

Figure 8: Model quality and latency for VE-variants. VE-full matches the best model performance of VE-lazy with less cumulative visible latency (shown with a log-scale).

Finally, we evaluate the sensitivity of the hyperparameters. We perform this analysis when measuring quality using the evaluation set rather than performing 3-fold validation over the labeled set in order to evaluate the behavior of feature selection under more ideal settings. We find that the quality is ≥95% for all datasets except BDD across a reasonable range of hyperparameter values ($w \in [3, 5, 7], C \in [5, 7], T \in [20, 50]$). BDD's selection correctness ranges from 0.68 to 0.88 for all settings. The evaluation set gives a more reliable estimate of feature quality, so the correct feature is picked even with less smoothing and a shorter time horizon.

5.4 Task scheduler

Finally, we evaluate the effectiveness of the optimizations described in Section 4 and show they enable VOCALExplore to match or exceed the model quality of VE-lazy but at a fraction of the visible latency. Figure 8 shows model quality and cumulative visible latency across 100 Explore steps (note that latency is shown with a log-scale). As in Section 5.1, we assume the user takes 10 seconds to watch and label each video clip. The VE-lazy variants perform feature and acquisition function selection as described in Section 3, but without the optimizations from Section 4. VE-lazy (PP) includes the preprocessing time to extract *all* candidate features from all videos, which is necessary because the ALM does not initially know the best feature. VE-lazy (X) variants perform incremental feature extraction as needed when the ALM switches to Coreset sampling. X indicates the number of unlabeled videos that have features extracted to serve as the candidates for Coreset. Larger X values have higher F1 on K20 (skew), and to a lesser extent on Deer, but the additional feature extraction tasks increase visible latency. Finally, VE-full, which uses all of the optimizations described in Section 4 matches or exceeds the F1 score achieved by the lazy variants but at much smaller visible latency (~1 second/step). VE-full exceeds the performance of the incremental variants on K20 (skew) because it extracts features from more videos in the background than the values of X we evaluated, so Coreset sampling performs better.

6 RELATED WORK

Video querying systems. Current video querying systems such as EVA [49], VIVA [38], and others [6, 10, 11, 24, 31] focus on efficient execution of queries over the outputs of pretrained models. Panorama [54] supports queries over novel labels using embedding similarity, but it focuses on recognition and verification rather than exploration and domain-specific model building.

Cloud vendor offerings. The large cloud computing vendors offer video analysis services [1, 2, 4] that automatically index videos with common objects, scenes, or activities. However, they do not support users in finding examples to label to train custom models.

Data exploration. Current video browsing systems [21, 39] are optimized for known-item search rather than exploration. Lancet [52] combines active learning and semi-supervised learning. While VOCALExplore uses pretrained models to extract embeddings for unlabeled data, Lancet jointly learns an embedding model and classifier. This is expensive because embeddings must be updated when the model is retrained. VQL [47] enables video exploration using the outputs of pretrained models, however it assumes the model is from the same domain as the target exploration. Forager [3] enables efficient exploration and domain-specific model training over images or individual video frames rather than video clips.

Zero-shot ML and unsupervised learning. Image-language models capable of zero-shot inference over images and text have recently proliferated, such as CLIP [37]. However, as we show in Section 5.3, embeddings from video models outperform CLIP on datasets where the labels cannot be determined by looking at a single frame, such as deer activity classification. Thus far there has been limited work to develop video-language models. VideoCLIP [48] is capable of zero-shot inference over videos, however it performs poorly on the datasets we evaluate on, achieving a macro F1 score of 0.04 for Deer and 0.33 for K20. Given the low zero-shot accuracy of current video-language models, it is necessary to implement domain-specific models. VOCALExplore could be extended to incorporate unsupervised learning to leverage the entire dataset, however current techniques for videos [19] require training a large model and repeatedly extracting updated feature representations from videos, which is too slow for our goal of an interactive system.

AutoML. VOCALExplore shares similarities with AutoML systems [20] as it supports training models by automatically selecting a feature extractor and sampling data. VOCALExplore, however, does not attempt to maximize model quality via techniques that traditionally fall under the umbrella of AutoML such as feature engineering [26, 46, 50], data augmentation [53], hyperparameter optimization [23], or model selection [27]. Instead, it rapidly produces an initial model with minimal user-perceived latency.

7 LIMITATIONS

VOCALExplore currently does not try to detect inconsistent labels that could arise from differences between users. However, we observe that even when randomly changing up to 20% of the labels, VOCALExplore achieves an F1 score similar to when all labels are correct. We include these results in our technical report [14]. This indicates that our techniques are robust to reasonable amounts of noise and still pick good feature representations.

While VOCALExplore is designed for activity classification tasks, it could be extended to support additional tasks like object detection. To do so, it would have to solve two ML tasks: region proposal and region classification. To obtain region proposals, VOCALExplore would have to decide between using a pretrained model vs. training its own with supervised data. Pretrained region proposal models will not work well if the objects in the video frames exhibit different "objectness" properties from what the model was trained on (e.g., objects in histopathology images have different properties than objects in wildlife images). Therefore, VOCALExplore would need to detect when regions proposed by a pretrained model are not of sufficient quality and in these case train a domain-specific region proposal model in addition to a domain-specific classification model. Once VOCALExplore can identify regions in frames that likely contain objects, then it becomes a classification task again. Instead of extracting features from entire frames, VOCALExplore would extract features from just the regions of interest.

While adding additional feature extractors would not be required, it may be beneficial to add models pretrained on object detection. The task scheduler would not need to be tuned because it dynamically schedules tasks based on observed latencies. Model training and user labeling latencies will likely change, but the TS will adapt by scheduling tasks more- or less-eagerly.

8 CONCLUSION

This paper presents VOCALExplore, a system that supports building domain-specific models over videos. VOCALExplore automatically determines how to select samples to be labeled and picks the best feature extractor for a given dataset. It implements optimizations to enable low-latency API calls while maintaining model quality.

ACKNOWLEDGMENTS

This work was supported in part by the NSF through awards CCF-1703051 and IIS-2211133 as well as a grant from CISCO. Thank you to A. Craig and A.J. Wirsing for providing the deer video data. Deer video data collection was carried out with permission, guidance, and logistical support from the Colville Tribes Fish and Wildlife Department, with special thanks to E. Krausz and R. Whitney, and under National Science Foundation grants DEB1145902 to A.J. Wirsing and DEB1145522 to M.R. Heithaus as well as UW IACUC Protocol #4226-01. Additional funding was provided by the Safari Club International Foundation, Conservation Northwest, the Washington Department of Fish and Wildlife (Aquatic Lands Enhancement Account (ALEA)), and the University of Washington Student Technology Fee (STF) program.

REFERENCES

[1] 2022. Amazon Rekognition. https://aws.amazon.com/rekognition/.

[2] 2022. Azure Video Indexer. https://learn.microsoft.com/en-us/azure/azure-video-indexer/video-indexer-overview.

[3] 2022. Forager: Rapid Data Exploration and Model Development. https://cs.stanford.edu/~fpoms/.

[4] 2022. Google Cloud Video Intelligence API. https://cloud.google.com/video-intelligence.

[5] 2023. NVIDIA DALI. https://developer.nvidia.com/dali.

[6] Michael R. Anderson, Michael J. Cafarella, Germán Ros, and Thomas F. Wenisch. 2019. Physical Representation-Based Predicate Optimization for a Visual Analytics Database. In *IEEE*. IEEE, 1466–1477.

[7] Favyen Bastani, Songtao He, Arjun Balasingam, Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan, Michael J. Cafarella, Tim Kraska, and Sam Madden. 2020. MIRIS: Fast Object Track Queries in Video. In *SIGMOD*. 1907–1921.

[8] Sara Beery, Guanhang Wu, Vivek Rathod, Ronny Votel, and Jonathan Huang. 2020. Context R-CNN: Long Term Temporal Context for Per-Camera Object Detection. In *CVPR*. Computer Vision Foundation / IEEE, 13072–13082.

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

[10] Jiashen Cao, Karan Sarkar, Ramyad Hadidi, Joy Arulraj, and Hyesoon Kim. 2022. FiGO: Fine-Grained Query Optimization in Video Analytics. In *SIGMOD*, Zachary Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 559–572.

[11] Yueting Chen, Xiaohui Yu, and Nick Koudas. 2022. Ranked Window Query Retrieval over Video Repositories. In *ICDE*. IEEE, 2776–2791.

[12] Gui Citovsky, Giulia DeSalvo, Claudio Gentile, Lazaros Karydas, Anand Rajagopalan, Afshin Rostamizadeh, and Sanjiv Kumar. 2021. Batch Active Learning at Scale. In *NeurIPS*. 11933–11944.

[13] 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*.

[14] Maureen Daum, Enhao Zhang, Dong He, Stephen Mussmann, Brandon Haynes, Ranjay Krishna, and Magdalena Balazinska. 2023. VOCALExplore: Pay-as-You-Go Video Data Exploration and Model Building. *CoRR* abs/2303.04068 (2023).

[15] Justin Dellinger, Carolyn Shores, Apryle Craig, Shannon Kachel, Michael Heithaus, William Ripple, and Aaron Wirsing. 2021. Predators reduce niche overlap between sympatric prey. *Oikos* (12 2021). https://doi.org/10.1111/oik.08628

[16] Haoqi Fan, Tullie Murrell, Heng Wang, Kalyan Vasudev Alwala, Yanghao Li, Yilei Li, Bo Xiong, Nikhila Ravi, Meng Li, Haichuan Yang, Jitendra Malik, Ross Girshick, Matt Feiszli, Aaron Adcock, Wan-Yen Lo, and Christoph Feichtenhofer. 2021. PyTorchVideo: A Deep Learning Library for Video Understanding. In *Proceedings of the 29th ACM International Conference on Multimedia*. https://pytorchvideo.org/.

[17] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. 2021. Multiscale Vision Transformers. In *ICCV*. IEEE, 6804–6815.

[18] Li Fei-Fei and Ranjay Krishna. 2022. Searching for computer vision north stars. *Daedalus* 151, 2 (2022), 85–99.

[19] Christoph Feichtenhofer, Haoqi Fan, Bo Xiong, Ross B. Girshick, and Kaiming He. 2021. A Large-Scale Study on Unsupervised Spatiotemporal Representation Learning. In *CVPR*. Computer Vision Foundation / IEEE, 3299–3309.

[20] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowl. Based Syst.* 212 (2021), 106622.

[21] Silvan Heller, Mahnaz Parian, Maurizio Pasquinelli, and Heiko Schuldt. 2020. Vitrivr-Explore: Guided Multimedia Collection Exploration for Ad-hoc Video Search. In *SISAP (Lecture Notes in Computer Science)*, Vol. 12440. Springer, 379–386.

[22] Silvan Heller, Loris Sauter, Heiko Schuldt, and Luca Rossetto. 2020. Multi-Stage Queries and Temporal Scoring in Vitrivr. In *ICME*. IEEE, 1–5.

[23] Kevin G. Jamieson and Ameet Talwalkar. 2016. Non-stochastic Best Arm Identification and Hyperparameter Optimization. In *AISTATS (JMLR)*, Arthur Gretton and Christian C. Robert (Eds.), Vol. 51. 240–248.

[24] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics. *PVLDB* 13, 4 (2019), 533–546.

[25] Siddharth Karamcheti, Ranjay Krishna, Li Fei-Fei, and Christopher D. Manning. 2021. Mind Your Outliers! Investigating the Negative Impact of Outliers on Active Learning for Visual Question Answering. In *Annual Meeting of the Association for Computational Linguistics*.

[26] Ambika Kaul, Saket Maheshwary, and Vikram Pudi. 2017. AutoLearn - Automated Feature Generation and Selection. In *ICDM*. IEEE Computer Society, 217–226.

[27] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. 2017. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *JMLR* 18 (2017), 25:1–25:5.

[28] Yang Li, Jiawei Jiang, Jinyang Gao, Yingxia Shao, Ce Zhang, and Bin Cui. 2020. Efficient Automatic CASH via Rising Bandits. In *AAAI*. AAAI Press, 4763–4771.

[29] Yanghao Li, Saining Xie, Xinlei Chen, Piotr Dollár, Kaiming He, and Ross B. Girshick. 2021. Benchmarking Detection Transfer Learning with Vision Transformers. *CoRR* abs/2111.11429 (2021).

[30] Zhicheng Liu and Jeffrey Heer. 2014. The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE Trans. Vis. Comput. Graph.* 20, 12 (2014), 2122–2131.

[31] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. 2018. Accelerating Machine Learning Inference with Probabilistic Predicates. In *SIGMOD*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 1493–1508.

[32] TorchVision maintainers and contributors. 2016. *TorchVision: PyTorch's Computer Vision library*. https://github.com/pytorch/vision

[33] Oscar R. Moll, Favyen Bastani, Sam Madden, Mike Stonebraker, Vijay Gadepally, and Tim Kraska. 2022. ExSample: Efficient Searches on Video Repositories through Adaptive Sampling. In *ICDE*. IEEE, 2956–2968.

[34] Albert Orriols-Puig and Ester Bernadó-Mansilla. 2009. Evolutionary rule-based systems for imbalanced data sets. *Soft Comput.* 13, 3 (2009), 213–225.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary 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 *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[36] Mark Raasveldt and Hannes Muehleisen. [n.d.]. *DuckDB*. https://github.com/duckdb/duckdb

[37] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *ICML*, Vol. 139. PMLR, 8748–8763.

[38] Francisco Romero, Johann Hauswald, Aditi Partap, Daniel Kang, Matei Zaharia, and Christos Kozyrakis. 2022. Optimizing Video Analytics with Declarative Model Relationships. *PVLDB* 16, 3 (2022), 447–460.

[39] Luca Rossetto, Ivan Giangreco, and Heiko Schuldt. 2014. Cineast: A Multi-feature Sketch-Based Video Retrieval Engine. In *ISM*. IEEE, 18–23.

[40] Fritz W Scholz and Michael A Stephens. 1987. K-sample Anderson–Darling tests. *J. Amer. Statist. Assoc.* 82, 399 (1987), 918–924.

[41] Ozan Sener and Silvio Savarese. 2018. Active Learning for Convolutional Neural Networks: A Core-Set Approach. In *ICLR*.

[42] Burr Settles. 2009. Active Learning Literature Survey.

[43] Gunnar A. Sigurdsson, Gül Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. 2016. Hollywood in Homes: Crowdsourcing Data Collection for Activity Understanding. In *ECCV (Lecture Notes in Computer Science)*, Vol. 9905. Springer, 510–526.

[44] Lucas Smaira, João Carreira, Eric Noland, Ellen Clancy, Amy Wu, and Andrew Zisserman. 2020. A Short Note on the Kinetics-700-2020 Human Action Dataset. *CoRR* abs/2010.10864 (2020).

[45] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. 2018. A Closer Look at Spatiotemporal Convolutions for Action Recognition. In *CVPR*. IEEE, 6450–6459.

[46] Jonathan Waring, Charlotta Lindvall, and Renato Umeton. 2020. Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artif. Intell. Medicine* 104 (2020), 101822.

[47] Yifan Wu, Steven Mark Drucker, Matthai Philipose, and Lenin Ravindranath. 2018. Querying Videos Using DNN Generated Labels. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD*, Carsten Binnig, Juliana Freire, and Eugene Wu (Eds.). ACM, 6:1–6:6.

[48] Hu Xu, Gargi Ghosh, Po-Yao Huang, Dmytro Okhonko, Armen Aghajanyan, Florian Metze, Luke Zettlemoyer, and Christoph Feichtenhofer. 2021. VideoCLIP: Contrastive Pre-training for Zero-shot Video-Text Understanding. In *EMNLP*. 6787–6800.

[49] Zhuangdi Xu, Gaurav Tarlok Kakkar, Joy Arulraj, and Umakishore Ramachandran. 2022. EVA: A Symbolic Approach to Accelerating Exploratory Video Analytics with Materialized Views. In *SIGMOD*. ACM, 602–616.

[50] Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. 2018. Taking human out of learning applications: A survey on automated machine learning. *arXiv:1810.13306* (2018).

[51] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. 2020. BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[52] Huayi Zhang, Lei Cao, Samuel Madden, and Elke A. Rundensteiner. 2021. LANCET: Labeling Complex Data at Scale. *PVLDB* 14, 11 (2021), 2154–2166.

[53] Weihang Zhang, Yuma Kinoshita, and Hitoshi Kiya. 2020. Image-Enhancement-Based Data Augmentation for Improving Deep Learning in Image Classification Problem. In *ICCE-TW*. IEEE, 1–2.

[54] Yuhao Zhang and Arun Kumar. 2019. Panorama: A Data System for Unbounded Vocabulary Querying over Video. *PVLDB* 13, 4 (2019), 477–491.