Coping With Data Drift in Online Video Analytics

Ioannis Xarchakos
University of Toronto
Toronto, Canada
xarchakos@cs.toronto.edu

Nick Koudas
University of Toronto
Toronto, Canada
koudas@cs.toronto.edu

ABSTRACT

Recent advances in Deep Learning and Computer Vision have enabled sophisticated image and video understanding. Dealing with real video streams however, posses various challenges such as coping with factors that change the video data distributions such as varying camera angles. When the video stream distribution changes, models that we have been using to process the video stream may no longer be applicable.

In this paper we present algorithms to monitor a video stream and detect when the underlying data distribution has changed in a light weight manner. The basis of our proposal are conformal martingales that can efficiently construct an understanding of the current video stream and detect changes in it very efficiently. We present the Drift Inspector Algorithm (DI) that encompasses such martingales to detect changes in the video distribution efficiently. We then propose two algorithms, namely Model Selection Based on Output (MSBO) and Model Selection Based on Input (MSBI) to efficiently select new models to continue processing the video stream when the distribution has changed.

We present the results of our evaluation involving various real video streams and analyze the trade-offs and the relative merits of our approach. Compared with other applicable state of the art approaches our proposals achieve up to an order of magnitude performance savings while being superior in their resulting accuracy in practical query scenarios.

1 INTRODUCTION

Recent advances in Deep Learning (DL), have revolutionized many computer vision and natural language processing applications. Several state-of-the-art computer vision algorithms have been proposed in the areas of image classification, object detection, and tracking [20, 38, 62]. This rapid progress has influenced the development of several video analytics systems [4, 15, 29, 32, 33, 37]. The basic premise of these systems is to enable a declarative query framework on top of videos (streaming or offline) making video content fully accessible to SQL-style query processing. As a result video objects, their relationships, actions and spatio-temporal constraints are fully available for querying. In addition, these systems reduce the per-frame processing time cost and achieve solid accuracy on par with state-of-the-art computer vision algorithms.

One fundamental drawback of state-of-the-art deep learning video analytics algorithms is the assumption that the distribution of data they operate on (serving data) is the same as the distribution of the data they have been trained on (training data). When serving data diverges from training data, the accuracy and effectiveness of video analytics algorithms degrade substantially [64].

Such divergence between serving and training data is known in the literature as *data drift* (or data shift) [56] and it has numerous manifestations (e.g., Covariate Shift, Concept Shift [44]). In video analytics, it occurs naturally and it is very common. Consider for example a surveillance application consuming data from a fixed camera to detect certain objects (e.g., cars on a highway). The underlying detection model [25] for cars has been trained utilizing repositories of car data or videos. In a production environment, the natural conditions will inevitably drift away from those in training data. For example, the models will have to operate under night conditions, fog, snow, rain etc. Likely, the underlying model has not been trained on data under all those adverse conditions (signifying data drift) and as a result, performance will deteriorate [64]. In video processing, drifts can be attributed to other factors. For example, changes in the camera angle as well as whether the camera is static or moving may impact model accuracy significantly depending on how substantial the changes in angle and movement speed are.

Data drift is a real challenge for video analytics as it is difficult to anticipate all possible conditions under which drift may occur. As such the main focus is on detecting data drift and when needed adjusting the underlying models to correct their performance in light of the new operational conditions.

The work closest to ours is ODIN [64]. It initiates work in drift detection and recovery in video analytics. ODIN consists of 3 modules; ODIN-Detect (detects whether a drift has occurred), ODIN-Select (selects models to process input frames), and ODIN-Specialize (trains a new model). As frames arrive, ODIN clusters them to an existing cluster or creates a temporary one. The process of making a temporary cluster permanent (signifying drift detection) is slow (Section 6.1). Furthermore, ODIN clusters each frame and selects the model associated with the cluster, slowing down the overall model selection process. This process is driven by a distance metric, resulting in poor selection performance (Section 6.2). To improve it, ODIN allows frames to fall into multiple clusters creating ensemble models for a frame. This affects the end-to-end time performance (more models are invoked for a frame), and the query accuracy (the ensemble model is inferior to the single best model (Section 6.3)).

We design a general architecture for operational video analytics coping with data drift. We focus on an operational environment where data drift may occur. To design a general solution, it is natural to assume that some adverse operational conditions are anticipated (e.g., detecting objects at night, under rain or snow) and suitable models have been provisioned and trained with data corresponding to such conditions. However, the operational environment has to cope with unknown conditions with no applicable models at our disposal. As such any general solution has to be equipped with the ability to *detect* data drift. Once a data drift is detected, to cope with the data distribution changes a best-performing model suitable for the new data has to be *selected* among a set of already provisioned models. If no such model can be identified, this will signify that the data has

drifted away from previously identified distributions. As such, a suitable model has to be provisioned for the new data.

In this paper, we propose algorithms to detect data drift by monitoring the video stream in a lightweight manner. Our proposed algorithm, called *Drift Inspector* (DI) is able to incrementally monitor the video stream and assess whether the underlying video frame distribution signifies a substantial change. We utilize principles of conformal theory [69–71] for this purpose and we demonstrate that our approach offers state of the art drift detection compared to other approaches with substantial performance benefits.

Once a drift is identified, we examine a set of available models in our disposal to identify whether one of the models is suitable to process the new data with high accuracy. We propose two algorithms for this purpose. The first, called Model Selection Based on Input (*MSBI*), examines the new incoming data after the data drift and compares them in a formal manner to the training data of existing models, to determine whether any of the existing models is suitable to process the new input stream. The second, called Model Selection Based on Output (*MSBO*) organizes the models in a suitable pre-processing step to quantify the *uncertainty* in a formal manner [40, 68] of the model output (its predictions) given a new input. The algorithms will promote a model they determine is applicable to continue processing the video without disruptions as a result of the drift. If no such model can be identified from the set of applicable models, the algorithms signify that a novel input has been identified for which a new model has to be constructed. An overview of the architecture is presented in Figure 1. Video frames are routed to algorithm DI and processed by the deployed model when no drift is detected. Once a drift is detected, a new model is selected, either from the list of existing models or by building a new model. In either case, a new model is deployed for further processing. Specifically, we make the following contributions:

- We present the *DI* algorithm, which is based on conformal prediction theory. The algorithm, using *conformal martingales* [70], formally and efficiently detects changes in the underlying frame distribution of a video stream, pinpointing the exact frame where the drift occurs. While conformal martingales have been utilized before for change point detection in other domains [34, 47, 67], we apply them in video streams for the first time. To achieve that, we make the following contributions. We develop a variational autoencoder to generate i.i.d samples (Section 4.2.2). Next, we design a custom betting function (Section 4.2.3) as well as a statistical test to pinpoint drift in video streams (Section 4.2.5).
- We present 2 algorithms, namely MSBI and MSBO, to recover from drift in video streams. MSBI algorithm is grounded on conformal martingales to evaluate new data arriving as part of the data drift against training data on our existing models. MSBO quantifies the predictive uncertainty of the existing models given the new data. We evaluate the relative tradeoffs between these algorithms and analyze the conditions under which they perform best.
- We present the results of a comprehensive performance evaluation using real data sets analyzing each proposed algorithm in isolation for its suitability and effectiveness for the associated task compared to state-of-the-art approaches. We also benchmark the performance of our end-to-end solution combining all of our proposals into a single system, comparing it to alternative solutions. Our results indicate that our techniques constitute a robust offering demonstrating significant drift detection and model selection performance benefits. Our combined proposals are 3 times faster and achieve 20% higher query accuracy than the state-of-the-art.

This paper is organized as follows: In Section 2 we review the related work. Section 3 presents the problem statement of this work, Section 4 presents the drift detection algorithm, followed by Section 5 which demonstrates the process of selecting a new model to process video frames captured after the change point detection. Section 6 presents our experimental evaluation. Section 7 concludes the paper discussing avenues to future work in this area.

2 RELATED WORK

Acknowledging the significance of query processing on streaming video, several recent works focus on declarative query processing over video stream [11, 13, 32, 33, 37, 39, 48, 59, 76, 77]. Noscope [33] focuses on fast query processing, on binary frame classification, by identifying queried object classes in video frames. SVQ [37, 74] presents a declarative query processing framework that supports query predicates involving spatial and aggregate constraints among objects on video streams while BlazeIt [32] supports aggregate queries. Haynes et. al [23, 24] present a visual data management system and discuss storage aspects of video query processing. Bastani et. al [4, 5] use object tracking to answer count and spatial-constrained queries. Several systems have proposed [29, 45] query processing on video streams while discussing scalability, resource utilization, and storage aspects. Several systems propose declarative query processing frameworks that support human-object interaction predicates [10, 12, 75]. Chen et. al [14–16] discuss aspects of temporal query processing on video streams.

Data drift in Machine Learning and its implications for learning algorithms have been studied in the past [6, 56, 57]. Various forms for changes in the underlying data distribution to learning have been studied and these include *concept shift*, *covariate shift*, and *prior probability shift*. The techniques we present apply both to problems related to concept shift and covariate shift. For a recent survey of various notions of distribution change and the relationship to learning algorithms, see [44]. Overall approaches in the literature can be classified as follows. First, in the context of drift detection, traditional approaches include control charts [53, 60] which require full knowledge of the distribution. Although they provide a formal framework they are not applicable in practice as we do not have parametric models of real video streams. In statistics [3] there are numerous parametric tests to quantify distribution change but again require knowledge of distributions in analytic closed forms. Non-parametric tests from statistics [9] such as Kolmogorov-Smirnoff (KS) [31] are robust statistical tests that can be applied for distribution change as well. In particular, two-sample KS tests can decide whether two samples come from the same distribution or not. However, although very efficient in one dimension, multidimensional KS tests are not efficient to compute [31] and closed-form results are not available. Other works have used unsupervised methods, such as clustering, to detect drift [64],[63] with ODIN [42] being the state of the art for which we present a detailed evaluation of our proposal against. Ekya [1] recovers from data drift in streaming videos by regularly retraining the deployed models on edge

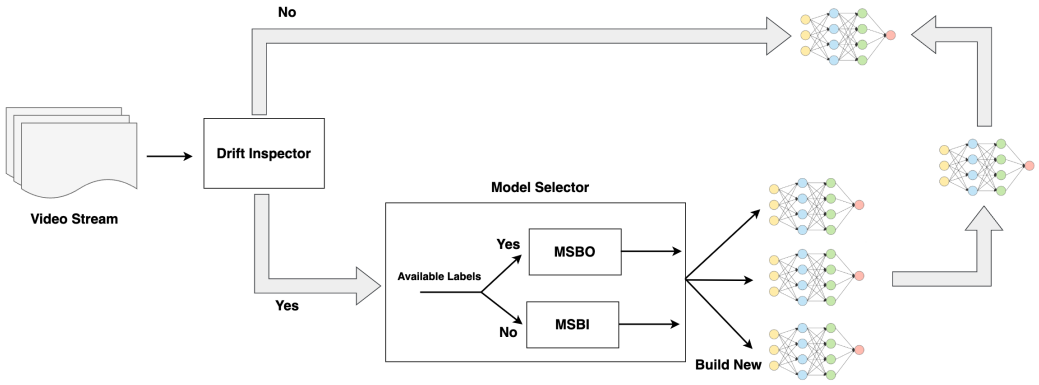


Figure 1: Example of the overall architecture

servers while Nagar et. al [50] utilize drift detection to solve the problem of temporal video segmentation of daylong egocentric video streams.

Martingales is a well-known concept in statistics [70] with numerous applications in statistical analysis. Conformal Martingales have been proposed in the context of conformal predictions [69–71]. Optimal stopping rules for drift detection using martingales have been studied in the past [49] however these results are of theoretical interest as full knowledge of all distributions and their parametric forms is assumed. Various methods, such as Transcend [30], employ martingales to detect drift in security domains [34, 47, 67].

Traditionally, Bayesian neural networks provide ways to measure the predictive uncertainty of a model [2, 19]. Despite the advances of Bayesian neural networks, they require significant assumptions, alterations in the training procedure, and their computational requirements make them unrealistic. A variety of Bayesian approximations have been proposed including Markov chain Monte Carlo [51] and their variations [36], variational Bayesian methods [43, 73] and Monte-Carlo Dropout [18]. Approaches for uncertainty quantification for predictive models, utilizing deep networks [2, 19, 27, 40] have been demonstrated to be more effective in practice [52]. Although ensembles of deep models have been utilized for uncertainty quantification [2, 40], we are not aware of any work utilizing deep ensembles for model selection.

3 PROBLEM STATEMENT

A video stream is an unbounded sequence of frames $S = \{f_1, f_2, f_3 \ldots\}$, where each f_i is a multidimensional vector. Assume that $f_1, f_2, \ldots, f_\theta \sim \mathcal{F}_k$, where $\mathcal{F}_k$ represents the underlying data distribution from which frames $f_1 \ldots f_\theta$ are drawn. From a practical point of view, $\mathcal{F}_k$ represents the distribution of the frames of a segment in the video stream for which a provisioned deep model M_k is utilized delivering satisfactory accuracy (e.g., detecting a collection of object classes). Let $f_{\theta+1}, f_{\theta+2}, \ldots \sim \mathcal{F}_{k+1}$ express the new underlying data distribution occurring at $\theta > 1$ as a result of a data drift in the video stream. Both $\mathcal{F}_k, \mathcal{F}_{k+1}$ are unknown distributions and we can only sample from them.

The first problem of interest in our work is to detect the *data drift* at time θ and declare that the distribution has changed, in a lightweight manner by monitoring (sampling) the video stream. We will propose the *Drift Inspector* algorithm in Section 4 for this purpose. Once a data drift is detected we wish to identify whether a model at our disposal in a collection of $M_i, 1 \leq i \leq m$ models can be utilized to process the new data. We will propose the *Model Selector* algorithm (MS) in Section 5. It is possible that none of the m models is applicable to process the new data and as a result, the MS algorithm will flag the necessity to create a new model.

In Section 6, we evaluate the effectiveness of our proposals and compare them against ODIN [64], a state-of-the-art drift detection and recovery system in video analytics. In evaluating drift detection performance, we introduce a metric quantifying the number of frames processed before detecting a drift. Both Drift Inspector and ODIN-Detect exhibit the ability to accurately identify drifts while avoiding false positive detections. Hence, our assessment focuses on their efficacy in promptly identifying drifts from the moment the distribution changes, providing insights into their responsiveness to evolving video stream dynamics.

The Model Selector (MS) algorithm, chooses a single model to process all frames following a drift, contrasting with ODIN-Select's approach of selecting a model (or ensemble) for each frame. The MS algorithm always selects the single best model to process incoming frames. To assess MS against ODIN-Select, we introduce a metric called, "model invocations per frame". This metric indicates the number of models invoked to process a single frame. MS always selects one model per frame, resulting in model invocations equal to processed frames. Conversely, ODIN-Select often forms an ensemble, yielding more than one model invocation per frame. This metric is crucial for evaluating end-to-end time performance and accuracy. More model invocations increase processing time, while ensembles reduce query accuracy compared to selecting the single best model. Finally, we assess the time overhead introduced by our proposed algorithms relative to ODIN during video processing.

4 DRIFT INSPECTOR

We will utilize principles from conformal theory [69–71] to design the *Drift Inspector* algorithm. Let $S = f_1, \ldots f_{n-1}$ a frame sequence. We are interested to develop measures that quantify how "strange" f_n is with respect to S. We will use the conformal prediction framework [70] for this purpose. We define a *nonconformity measure* as a function mapping $(f_n, S) \rightarrow C(f_n, S)$, where $C(f_n, S)$ is a real number with the following semantics: the greater the value is the stranger f_n is with respect to S. Numerous measures can be utilized to define this mapping. For example, we can consider the nearest neighbor conformity measure and quantify C as the average distance of f_n to the members of S.

Table 1: Notation table

$\mathcal{T}_i$	Training data for distribution i
$A_{\mathcal{T}_i}$	The variational autoencoder for distribution i
$\Sigma_{\mathcal{T}_i}$	The i.i.d samples generated by $A_{\mathcal{T}_i}$ from $\mathcal{T}_i$
M_i	A model that processes frames coming from distribution i
A_i	The non-conformity score of each element in $\Sigma_{\mathcal{T}_i}$
f	A single video frame
$f_1, \ldots f_n$	A sequence of video frames
a_f	The non-conformity score of frame f
g	A betting function
S	The conformal martingale scores of W-1 past observations
K	The number of nearest neighbours
W	A window of observations
r	The significance level

Table 1 presents a summary of the notation we use in Sections 4 and 5.

Having established a non-conformity measure, the next step in conformal predictions is to define the p-value for a new observation f_n as $p_n = p(f_1, \ldots, f_n) =$

$$\frac{|\{i = 1, \ldots n : \alpha_i > \alpha_n\}| + U|\{i = 1, \ldots, n : \alpha_i = \alpha_n\}|}{n} \quad (1)$$

where U is a uniform number in $[0, 1]$ and a_i the non-conformity measure defined as:

$$a_i = C(f_i, \{f_1 \ldots f_{i-1}, f_{i+1}, \ldots f_n\}) \quad (2)$$

Thus, the p-value of f_i is defined as the fraction of frames that have a non-conformity score greater than or equal to the non-conformity score of α_i. Intuitively the smaller p-value is for a frame the stranger it is with respect to frames S. A fundamental theorem in conformal predictions [70] is the following:

Theorem 4.1. *[71] If $f_1, \ldots f_n$ are independent and identically distributed (i.i.d), then the p-values $p_1, p_2, \ldots$ are independent and uniformly distributed in [0,1]*

As a corollary, if $f_1, \ldots f_\theta$ are indeed i.i.d according to $\mathcal{F}_k$ and the distribution changes at $f_{\theta+1}, \ldots$ to $\mathcal{F}_{k+1}$, the corresponding p-values will not be i.i.d uniform in [0,1]. We will use this insight in the sequel to anchor our approach.

4.1 Exchangeable Martingales

Let $x_1, x_2, \ldots$ be a sequence of random variables in R^d. The joint probability distribution of $x_1, \ldots x_N$ for a finite N is exchangeable if it is invariant for any permutation of these random vectors. Every exchangeable distribution is a mixture of i.i.d distributions [69].

An exchangeability martingale is a sequence of non-negative random variables $\mathcal{S}_1, \mathcal{S}_2, \ldots$ such that

$$\mathbb{E}(\mathcal{S}_{n+1}|\mathcal{S}_n, \ldots \mathcal{S}_1) = \mathcal{S}_n, n = 1, 2, 3, \ldots \quad (3)$$

where $\mathbb{E}$ is the expectation with respect to any exchangeable distribution. According to Vovk et. al [70] in this case:

$$\mathbb{P}(\exists n : \mathcal{S}_n > c) \leq \frac{1}{c}, c \in \mathbb{R} \text{ and } c \geq 1 \quad (4)$$

for any exchangeable distribution. Equation 4 provides a way to test for i.i.d (equivalently exchangeability). If the final value of $\mathcal{S}_n$ is large, we can reject the i.i.d (or exchangeability) assumption with the corresponding probability. We will use this observation with p-values.

Given a sequence of p-values, consider the martingale:

$$\mathcal{S}_n = \prod_{i=1}^{n} g_i(p_i), n = 1, 2, \ldots \quad (5)$$

where each $g_i(p_i) = g_i(p_i|p_1, \ldots, p_{i-1})$ is a function (also called betting function [69]) satisfying $\int_0^1 g_i(p)dp = 1$. The intuition behind a betting function is that for small p-values, namely when the corresponding new observation is not strange compared to the existing values, we want the g_i to return a small value. For larger values of p_i however, which signifies that the new observation is strange compared to the existing values, we like to penalize it for having the corresponding g_i return high values and consequently detect that the p values are not uniformly distributed. We can verify the martingale property under an exchangeable distribution:

$$\mathbb{E}(\mathcal{S}_{n+1}|\mathcal{S}_n, \ldots \mathcal{S}_1) = \int_0^1 (\prod_{i=1}^{n} g_i(p_i)) g_{n+1}(p)dp =$$
$$(\prod_{i=1}^{n} g_i(p_i)) \int_0^1 g_{n+1}(p)dp = \prod_{i=1}^{n} g_i(p_i) = \mathcal{S}_n \quad (6)$$

There are multiple *betting functions* with desirable properties in the literature [69] that can be adopted for this purpose.

4.2 Martingales for Drift Detection

In order to develop a workable solution using martingales for drift detection we need to a) overcome the limitation that video frames in a frame sequence are not independent as we expect correlations from one frame to the next in a stream b) specify efficient ways to compute p-values on video frames.

4.2.1 Independent Video Frames. A video frame sequence is expected to exhibit correlations across frames. However, a fundamental requirement in computing p-values in a way to guarantee their properties is that the observed values are independent and identically distributed. In our setting, we assume that m models have been provisioned and are available to process the video stream. Thus, we also assume that along with each model $M_i, 1 \leq i \leq m$ we also have access to the associated training data $\mathcal{T}_i$. We use each $\mathcal{T}_i$ to construct a VAE $A_{\mathcal{T}_i}$ [35, 58]. The VAE provides a way to sample in an i.i.d manner from the underlying distribution of $\mathcal{T}_i$. Once $A_{\mathcal{T}_i}$ is constructed, we can generate any number of samples from the distribution $\mathcal{T}_i$ was generated from. We will use $\Sigma_{\mathcal{T}_i}$ to designate the set of samples generated from $\mathcal{T}_i$ using $A_{\mathcal{T}_i}$.

4.2.2 Variational Autoencoder (VAE). To generate i.i.d samples $\Sigma_{\mathcal{T}_i}$, we construct a VAE. The VAE does not require re-training once it is trained for some distribution i. Although we train the VAE on the input frames which exhibit correlations, in practise this does not affect our results. More elaborate VAE models that take correlations into account are available if desired [66]. We present its architecture in Figure 2. The encoder consists of 3 convolutional layers and 2 fully connected (FC) layers to optimize frame processing time. Extensive experimentation has revealed that this architecture is sufficient for generating latent representations that capture essential information from the input frames. Both FC layers take as input the output of the convolutional layers. One FC layer computes the standard deviation vector, while the other estimates the mean vector. The mean and standard deviation are learned during the training of the VAE on the training data $\mathcal{T}_i$. At the end of the encoder stage, we randomly sample the

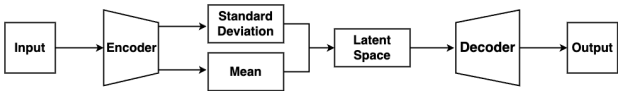


Figure 2: Autoencoder Architecture

Normal distribution using the learned mean and standard deviation to obtain i.i.d samples to form $\Sigma_{\mathcal{T}_i}$. The decoder consists of a single FC layer that takes the computed latent representation of the input frame as input, followed by 3 convolutional layers. The convolutional layers are responsible for reconstructing the input frame. To optimize the VAE, we employ a loss function that consists of two parts; the reconstruction error which maximizes the similarity of the reconstructed image, and the KL divergence to avoid overfitting. For the reconstruction error, we use the binary cross-entropy loss function which estimates the pixel-to-pixel difference between the input and the output.

4.2.3 Computing p-values. For a specific $\Sigma_{\mathcal{T}_i}$, upon a new observation f we compute $\alpha_f = C(f, \Sigma_{\mathcal{T}_i})$. The a_i value for each element of $\Sigma_{\mathcal{T}_i}$ is precomputed as all elements are available in advance. In terms of non-conformity measure, any score that measures the distance between two images [46] can be used to compute the non-conformity score of an incoming frame. We adopt the average Euclidean distance between f and elements of $\Sigma_{\mathcal{T}_i}$. That way, each p-value for every incoming frame f is computed incrementally using equations 1,2 requiring only the computation of the value α_f.

A potential drawback of using martingales of p-values in equation 5 in practice is that when the p-values are indeed i.i.d, the corresponding martingale being a product of betting functions will assume very small values. Thus when observations with larger p-values appear, signifying non i.i.d observations, the product may take time to increase in order to detect the drift at the suitable significance level (as per equation 4).

4.2.4 Constructing a custom betting function. To address this issue, we design a betting function to accelerate the drift detection [69]. Specifically, we take the logarithm on both sides of equation 5:

$$\log(\mathcal{S}_n) = \sum_{i=1}^{n} \log(g_i(p_i)), n = 1, 2, 3 \ldots \tag{7}$$

We would like $\log(\mathcal{S}_n)_{n=1}^{\infty}$ to be a martingale sequence:

$$\int_0^1 \log(g_i(p))dp = 0, i = 1, 2, 3, \ldots \tag{8}$$

It is easy to see that if we adopt the betting function properties in the case of multiplicative martingales (as in equation 5) namely $\int_0^1 g_i(p)dp = 1$, since the log() function is concave:

$$\int_0^1 \log(g_i(p))dp \leq log \int_0^1 g_i(p)dp = 1 \neq 0 \tag{9}$$

which does not imply the desired equation 8. To resolve this issue and obtain a valid martingale, we enforce the betting functions to integrate to zero. Thus:

$$\mathcal{S}_n = \sum_{i=1}^{n} g_i(p_i) \; with \int_0^1 g_i(p)dp = 0 \tag{10}$$

In this case, we have:

$$\mathbb{E}(\mathcal{S}_{n+1}|\mathcal{S}_n, \ldots \mathcal{S}_1) = \int_0^1 (\sum_{i=1}^{n} g_i(p_i) + g_{n+1}(p_{n+1}))dp_{n+1} = \sum_{i=1}^{n} g_i(p_i) + \int_0^1 g_{n+1}(p)dp = \mathcal{S}_n \tag{11}$$

thus we get a valid martingale in the additive case as well. In terms of betting functions that satisfy this property, one has numerous choices. For example an odd function $f(p)$: [-1/2,1/2] $\rightarrow$ R satisfies:

$$\int_{-1/2}^{1/2} f(p)dp = 0 \; implies \int_0^1 f(p - 1/2)dp = 0 \tag{12}$$

Thus, $g(p) = f(p - 1/2)$ is a valid betting function for any odd $f(p)$ (for example, $f(p) = -p$).

4.2.5 Testing for Drift Detection. Having designed proper additive martingales, we will create a statistical test that allows us with a certain probability to declare the drift, by monitoring the values of the martingale. The idea is that when the p-values are small and no drift exists, the values of the martingale will be bounded in a certain region with a high probability. Once the values exit this region a drift has occurred.

From the Hoeffding-Azuma [28] inequality, we know that for i.i.d random variables $y_1, \ldots y_l$ with $a_i \leq Y_i \leq b_i$, for any $t \geq 0$,

$$P(|\sum_{i=1}^{l} Y_i| \geq t) \leq 2exp(-\frac{t^2}{\sum_{i=1}^{l}(b_i - a_i)^2}) \tag{13}$$

In the additive martingale case, when the betting function is a shifted odd function with $|q(p)| \leq 1$, equation 13 becomes:

$$P(|\mathcal{S}_l| \geq t) \leq 2exp(-\frac{t^2}{2l}) \tag{14}$$

where $\mathcal{S}_l = g(p_1) + \ldots g(p_l)$. This can readily be used to design a statistical test for the drift. More specifically for a significance level r when $|\mathcal{S}_l| > \sqrt{2l(\frac{2}{\mathsf{r}})}$ we can reject the hypothesis that p-values follow a uniform distribution in [0,1] and declare a drift. To make this test even more efficient, we can impose window semantics. More specifically for a window of observations W, at significance level r a drift is declared when:

$$|\mathcal{S}_l - \mathcal{S}_{l-W}| > \sqrt{2W(\frac{2}{\mathsf{r}})} \tag{15}$$

In the observation window W, we assess the rate of change of the martingale score, not drift occurrences within W.

4.3 Drift Inspector Algorithm

We present the *Drift Inspector* algorithm. The algorithm accepts as input f which is a new frame observation taken from the video stream at time t, an iterator $iter$, $\Sigma_{\mathcal{T}_i}$ that represents samples taken from the variational autoencoder trained on training data $\mathcal{T}_i$ corresponding to model M_i $1 \leq i \leq m$. A_i is a list containing the pre-computed non-conformity score of each element in $\Sigma_{\mathcal{T}_i}$. $\mathcal{S}$ is a list that contains the conformal martingale scores of the past $W-1$ observations, where $S[0] = 0$. W and r are parameters as per Section 4.2. K is the number of nearest neighbors we use to measure the non-conformity score. Finally, the algorithm outputs the updated list S and variable $drift$ determining if a drift has occurred while processing frames coming from a video stream. The algorithm is capable of detecting a drift when the distribution

Table 2: Train data

$\Sigma_{\mathcal{T}_i}$	[[2, 3], [3, 1], [-1, 0], [4,4], [2,2]]
A_i	[1.8, 2.3, 4, 2.71, 1.72]

of the incoming frames is sufficiently different than the known distribution.

The variable *score* is initialized in line 2, which is used to measure the p-value score of the input observation. In line 3, we calculate the non-conformity score of frame f identifying the K closest observations between f and $\Sigma_{\mathcal{T}_i}$. In lines 4 to 8 we iterate through A_i to access the non-conformity score of each element in $\Sigma_{\mathcal{T}_i}$. We increase the variable *score* by one when the non-conformity score of an element in $\Sigma_{\mathcal{T}_i}$ is higher than a_f. When the two non-conformity scores are equal, a random value, which ranges from 0 to 1, increases the *score* variable. In line 10, we insert the score for the conformal martingale for frame f in S, utilizing the betting function $g(p)$ on p-value p as per Section 4.2. Finally, the list S (the conformal martingale score of the past $W-1$ observations) and the $drift$ (a boolean value that determines if a data drift is detected) are returned by the *Drift Inspector* algorithm.

Algorithm 1 Drift Inspector algorithm

```
1 def DI(iter, f, Σ_{T_i}, A_i, S, W, r, K):
2   score = 0
3   a_f = C(f, Σ_{T_i}, K)
4   for j in range(0, len(A_i)):
5       if A_i[j] > a_f:
6           score += 1
7       elif A_i[j] == a_f:
8           score += random(0,1)
9    p = score/len(A_i)
10   S.append(max(0, S[-1] + log(g(p))))
11   drift = False
12   window = iter if W > iter else W
13   if |S[iter] − S[iter − window]| > sqrt(2W(2/r)):
14       drift = True
15   return S, drift
```

4.3.1 Example. In Table 2, we show 5 samples from the training data $\Sigma_{\mathcal{T}_i}$, and we measure their non-conformity score in A_i. The two lists are provided as input in every invocation of Algorithm 1. Table 3 contains 4 input observations captured from an unknown distribution, and provided to the algorithm one at a time. The values of the rest of the input parameters are W=2, r=0.5, and k=3. Table 4 presents the variable values of algorithm DI. Column a_f shows the non-conformity score and p shows the p-value scores of each observation f. Finally, under $\mathcal{S}[iter]$, we observe the martingale score of a frame, while column $drift$ determines if a drift has occurred. We only include two-dimensional feature matrices for simplicity.

In Table 2 the elements in $\Sigma_{\mathcal{T}_i}$, have similar non-conformity scores A_i since all come from the same distribution. The frames under column f come from the same distribution but differ from those of $\Sigma_{\mathcal{T}_i}$. In Algorithm 1, line 3, we measure the distance of an observation f from its K nearest neighbors in the $\Sigma_{\mathcal{T}_i}$. Since the f and the elements of $\Sigma_{\mathcal{T}_i}$ are from different distributions, we expect a high non-conformity score a_f. In lines 4-8, we compare

Table 3: Input frames

iter	f
1	[8, 6]
2	[9, 8]
3	[10, 7]
4	[6, 7]

Table 4: DI variables & outputs

iter	a_f	p	$\mathcal{S}[iter]$	drift
1	6.1	0	1.5	False
2	7.6	0	2.5	False
3	8.3	0	5.4	False
4	5.2	0	8.5	True

the non-conformity scores of the elements of A_i and a_f to identify cases where the score of an element of $\Sigma_{\mathcal{T}_i}$ is higher than a_f.

As we observe in Table 4, none of the frames has a non-conformity score a_f lower than any element of A_i list (Table 2). Thus, all frames f attain a p-value score of 0 (line 9). In line 10, we measure the conformal martingale score of each observation, while S[0]=0. Since the p-values of the frames are zero, the conformal martingale score increases after each iteration. Given a window of observations W=2 and a significance level r = 0.5, the right part of the inequality becomes 4. We observe that, when iter=4, we declare drift since $S[4] - S[4-2] = 6.5 > 4$.

5 MODEL SELECTOR

Upon detecting a drift, we would like to continue processing the incoming data by selecting a suitable model, among a collection of m available pre-trained models. If we determine that none of the available models is suitable we would like to alert that a new model needs to be constructed to process the incoming data which represents a totally new distribution we never experienced before.

In this section, we will propose two approaches for model selection. The first approach called *Model Selector Based on Input* (MSBI) aims to identify a suitable model by comparing new data (arriving after the drift is detected) with the i.i.d sample of the training data $\Sigma_{\mathcal{T}_i}$ we have in our disposal for each model $M_i, 1 \leq i \leq m$. The main idea is to use the proposed Drift Inspector (DI) algorithm to test at the same significance level whether the new data are sufficiently different from each of the $\Sigma_{\mathcal{T}_i}, 1 \leq i \leq m$. The second approach *Model Selector Based on Output* (MSBO) uses suitable pre-processed versions of all existing models M_i and observes their output, on the new data. It uses the output to reason about *how confident* each model is to classify its input.

5.1 The MSBI Algorithm

The MSBI algorithm utilizes the DI algorithm for all available models M_i. It is depicted as algorithm 2. Given a sequence of W_N of frames coming after the drift, MSBI uses DI for each $\Sigma_{\mathcal{T}_i}$ at a specified significance level r. If the new data are sufficiently different from all i.i.d samples existing models are trained on, we are confident that the new data come from a previously unseen distribution and a new model should be constructed. On the contrary, if the hypothesis cannot be rejected for a distribution i, we adopt model M_i to process the new data. If multiple models are suitable, we break ties arbitrarily or progressively by increasing

the significance level and repeating the test until only a model remains.

5.1.1 MSBI algorithm. We present the MSBI algorithm which selects a new model to process frames appearing after the drift, from a collection of models. First, the algorithm's input includes the window W_N with the collected video frames, and M which depicts the available models we utilize to select a model to process the video stream after the data drift. W and r are parameters as per Section 4.2 and K includes the number of nearest neighbors we use to measure the non-conformity score.

Algorithm 2 MSBI algorithm

```
1 def MSBI(W_N, M, W, r, K):
2   driftModel = dict()
3   for Σ_{T_i}, A_i, M_i in M:
4     S = list()
5     for iter, f in enumerate(W_N):
6       s, drift = DI(iter, f, Σ_{T_i}, A_i, S, W, a, K)
7       S.append(s)
8       driftModel[M_i] = drift
9   if all(driftModel.values()):
10     trainNewModel()
11  M = driftModel.values().index(False)
12  if len(M) == 1:
13     return M
14  MSBI(W_N, M, W, r+0.1, K)
```

In line 2, we initialize the variable $driftModel$, which is a dictionary that stores a boolean value that determines if drift occurred during the processing of the frames W_N for each model. In line 3, we iterate through all available models M, accessing $\Sigma_{\mathcal{T}_i}$, the A_i which is a list containing the pre-computed non-conformity score of each element in $\Sigma_{\mathcal{T}_i}$ and the model M_i. Next, we initialize the list S which stores the conformal martingale score of the W most recent frames. In line 5, we use the W_N frames as input to trigger the DI algorithm. Algorithm DI returns the conformal martingale score as well as the variable $drift$ which determines if a drift occurred. In line 8, we assign a boolean value for each model M_i, to determine if a drift occurred while using this model. After processing the frames for all models, in lines 9-10, we determine if a drift is detected for all the models in M. In this case, we construct a new model trained on the new data distribution. Alternatively, we determine the model with the value False, and we select it to process the video frames. When more than one model is assigned the value False, we trigger the MSBI algorithm. The MSBI algorithm takes as input a new set of models while the significance level r is increased.

5.2 The MSBO Algorithm

The MSBO algorithm adopts an approach complementary to that of MSBI. Instead of examining how similar the new data W_N are with the data utilized to train our models, it utilizes the model predictions to base its decision. Simply observing the score distribution of each model M_i on the data in W_N and computing various statistical moments [55] to conduct a comparison is not a viable approach. The reason is that it is well-known [22, 26, 41] that deep models appear overconfident (yield high sigmoid scores in their output) for out of distribution samples [26] which is exactly the scenario we wish to capture.

5.2.1 Proper scoring rules. We seek to preprocess the training data $\mathcal{T}_i$ each model M_i has been trained on, to be able to derive a measure of *confidence* of a model on its predictions. The basis of our approach is to train networks using proper scoring rules [21] for measuring the predictive uncertainty of each model. A scoring rule assigns a numerical score to a predictive distribution $p_\theta(y|x)$, rewarding better predictions over worse. We focus on rules that a higher numerical score is better. Let $F(p_\theta, (y, x))$ be a scoring rule that evaluates the quality of $p_\theta(y|x)$ compared to $y|x \sim q(y, x)$, where q is the true distribution of (y, x). The expected scoring rule is then $F(p_\theta, q) = \int q(y, x)F(p_\theta, (y, x))dydx$. A proper scoring rule is one where $F(p_\theta, q) \leq F(q, q)$ with equality if and only if $p_\theta(y|x) = q(y|x)$, for all p_θ and q. The function $F(p_\theta, q(y, x)) = -\log p_\theta(y|x)$, which when maximized with fixed parameter y is the negative log-likelihood loss function that deep networks are typically trained on, is a proper scoring rule. In the case of multi-class K-way classification, the popular softmax cross entropy loss is equivalent to the log-likelihood and is a proper scoring rule. Also $\frac{1}{K}\sum_{i=1}^{K}(\delta_{i=y} - p_\theta(y = i|x))^2$ which is known as the Brier score [8], minimizing the squared error between the predictive probability of a label and the one-hot encoding of the correct label, is a proper scoring rule as well. A Brier score of zero indicates that the network has complete certainty in its predictive probability; the higher the score is the more uncertain the predictions are.

5.2.2 Model uncertainty estimation. Our goal is to derive an estimate of the predictive uncertainty of each model of new input. We will estimate it using a bootstrap approach (the members of the ensemble model are trained on different samples of the original train set) [17]. We build an ensemble of L models (typical values of L are between 3 and 10 [40]) for each M_i using the training data $\mathcal{T}_i$ [27]. The ensemble model does not require re-training, once it is trained for a distribution i. Each ensemble model $M_{i,l}$, $1 \leq l \leq L$ and each model is trained end to end minimizing the softmax cross-entropy loss, over a randomized shuffle of the entire $\mathcal{T}_i$. It has been previously observed [40] that using the entire training data for ensembles of deep networks provides greater accuracy, than following typical bagging techniques [7] as in the case of more traditional (non-deep) models. The parameters $\theta_{i,l}$ for each ensemble model $M_{i,l}, 1 \leq i \leq m, 1 \leq l \leq L$ are initialized randomly.

We treat the ensemble as a uniformly-weighted mixture model. Thus, we essentially estimate the predictive uncertainty of the corresponding model using a bootstrap approach [17]. For the ensemble corresponding to model M_i we combine the predictions as $p_{\theta_i}(y|x) = \frac{1}{L}\sum_{i=1}^{L} p_{\theta_{i,l}}(y|x, \theta_{i,l})$, where $p_{\theta_{i,l}}$ is the prediction of $M_{i,l}$ (the l-th ensemble model of M_i). If desired, we can obtain confidence intervals for the predictive uncertainty of the model using the bootstrap as well [17] at the desired confidence level.

Upon drift detection, we accumulate a window W_T of frames, past the point of drift, to evaluate the predictive uncertainty utilizing the ensemble model of each M_i. We compute the average value of $p_{\theta_i}(y|x), \forall x \in W_T$ which quantifies the average predictive uncertainty of model M_i on the new input. We use the Brier score to estimate the predictive uncertainty of each model M_i. Alternatively, we could estimate the uncertainty using the log-likelihood. However, the models are trained by minimizing the cross-entropy loss, which is equivalent to the log-likelihood. Hence, we use the Brier score for an unbiased estimation of the uncertainty. We return the model i with the lowest predictive uncertainty which is below a threshold h. If no such model exists

this signifies that the input is sufficiently different from what we have encountered before and we build a new model.

To calibrate the threshold h we use the existing models and their associated training data $\mathcal{T}_i$ to establish a baseline for uncertainty h. Let $S_{\mathcal{T}_i}$ represent a random sample of $\mathcal{T}_i$ of fixed size. We utilize $S_{\mathcal{T}_i}$ to obtain the predictive uncertainty pc^i_j using the ensembles for all $j \neq i, 1 \leq j \leq m$. We then compute pc^i_{avg} which is the average uncertainty for predicting $S_{\mathcal{T}_i}$ using the ensembles for models $M_{j \neq i}$. We set h one standard deviation below the mean of $pc^i_{avg}, 1 \leq I \leq m$. Arguably h establishes a baseline for the predictive certainty to infer a model's suitability to process the new data.

5.2.3 MSBO algorithm. We present Algorithm 3, which selects a suitable model to process frames appearing after a data drift. The algorithm accepts as input the W_T which is the window of the collected video frames and the available models M that we utilize to select a model to process the video stream after the data drift. pc_{avg} is a dictionary containing the average uncertainty of all the models M, σ is the standard deviation and L is the number of models that each ensemble model $M_{i,l}$ consists of.

Algorithm 3 MSBO algorithm

```
1 def MSBO(W_T, M, pc_avg, σ, L):
2   brier = dict()
3   for i in range(1, len(M)):
4       brierScore = 0
5       for frame, label in W_T:
6           vote = 0
7           for l in range(1, L):
8               prediction = M_{i,l}(frame)
9               vote += prediction
10          totalVote = vote/L
11          brierScore += (totalVote - label)^2
12      brier[model] = brierScore/len(frames)
13  best = min(brier)
14  k = best.keys()
15  if best.values() <= pc_avg[k] − σ[k]:
16      return k
17  trainNewModel()
```

In line 2, we initialize the variable *brier*, which is a dictionary that stores the Brier score of each model. Next, in line 3, we iterate through all the available models M, and for each model we are accessing the selected frames and their associated ground truth, to identify a model to process the video stream after the data drift. In lines 8 and 9, we evaluate a frame using a single model M_i for L times, since we have trained each model L times, and we average each network's output to make a final prediction for the frame, in line 11. Then, we calculate the Brier score using the ensemble's prediction of a frame and its associated label. Finally, when we have processed all the frames for a single model we add the model's average Brier score in the *brier* dictionary. We follow the same process for all the models M. In line 14, we find the model with the lowest predictive uncertainty, by selecting the one with the lowest Brier score. In lines 15 to 17, we determine if the uncertainty of the model is below the $pc_{avg}[k]$ which is the average uncertainty of the selected model $M_k, 1 \leq k \leq m$. If the score of the model is below the threshold, the model is deployed, otherwise, we trigger the module that is responsible for training a new model.

5.3 MSBO-MSBI Trade-off

Algorithms MSBO and MSBI achieve the same model selection accuracy, with MSBO being 4 times faster than MSBI, making it the preferred choice for model selection. However, MSBO requires training an ensemble model for each distribution found in the stream, along with annotations generated by Mask R-CNN for the training samples. In contrast, MSBI is fully unsupervised, only requiring the invocation of the variational autoencoder and the estimation of the conformal martingale score, which do not require labeled samples. While annotations generated by Mask R-CNN are suitable for some applications (e.g., traffic monitoring), MSBI is preferred to MSBO when obtaining high-fidelity annotations for incoming frames is impossible (e.g., autonomous driving guidance).

5.4 Model Training

In previous sections, we presented two algorithms for selecting a model, from a pool of available models, to process incoming frames following a drift. When Algorithms MSBI and MSBO invoke the $trainNewModel()$, signifying that a novel distribution has caused the drift, a new model is constructed. In this scenario, we first collect a sufficient number of frames following the drift (typically 5K frames, which amounts to 3 minutes worth of video at 30 fps) and annotate them using Mask R-CNN [25]. We then proceed by training all necessary models required for supporting drift detection and recovery as well as query processing. Specifically, we train a VAE for Algorithms DI and MSBI, and classification models (according to the user-defined queries) for query processing and Algorithm MSBO. When no models are provided, we immediately collect the incoming frames and train the necessary models as explained above.

6 EXPERIMENTAL EVALUATION

In this Section, we present the results of a thorough experimental evaluation of our proposals. We use 3 different real datasets namely, **Berkeley Deep Drive (BDD)** [78], **Detrac** [72] and **Tokyo** [4]. The Tokyo dataset is created from 3 different video streams on the same road intersection in Tokyo. **BDD** is a road-driving dataset that consists of 100,000 frames captured under various conditions, namely, different time of the day (day and night), and different weather conditions (rain, snow, etc) from dashboard cameras. **Detrac** is a traffic dataset that consists of 100 distinct fixed-angle video sequences captured at different locations in China. Initially, the **BDD** dataset (1280x720) had different image dimensions than **Detrac** and **Tokyo** datasets (960x540). We pre-processed the BDD dataset to match the image dimension of Detrac and Tokyo. In Table 5, we present a description and key parameters of each dataset. We use Mask R-CNN [25] to annotate the datasets. The datasets contain inherent data drifts. BDD provides labels to separate frames under different conditions (night, rain etc). For the Detrac and Tokyo datasets, we identify drifts by splitting the frames according to the camera angles they belong to. In the BDD dataset 4 drifts occur (switching to Night, Rain, Snow, and Day sequences), in Tokyo there are 3 drifts (switching to Angle 1, Angle 2, Angle 3), while the Detrac dataset contains 5 drifts (switching to Angle 1, Angle 2, ..., Angle 5).

We use Python 3.8 and the PyTorch framework [54] to implement our proposals on an HP workstation using 2 Nvidia Titan XP GPUs. We use 5K frames (i.e. 3 minutes worth of video frames at 30 fps) to train a VAE for each distribution i, for all datasets. We use horizontal flipping and random cropping [61]

Table 5: Datasets and their characteristics

Dataset	#Sequences	Stream Size	Obj/Frame	std
BDD	4	80,000	9.2	6.4
Detrac	5	30,000	17.2	7.1
Tokyo	3	45,000	19.2	4.7

Table 6: Drift detection time performance (in seconds)

Dataset	Drift Inspector	ODIN-Detect
BDD	293.4	636.2
Detrac	97.3	235.8
Tokyo	194.8	294

to increase the training size to 20k frames. We train our models with a batch size of 16 images per GPU using the Adam optimizer. The VAE employs binary cross-entropy, while the image classifiers use negative log-likelihood loss. For algorithms, Drift Inspector (DI) and Model Selection Based on Input (MSBI), the only component that requires training is the VAE. The VAE is trained once when a new distribution is detected. The training of the VAE takes approximately 1 hour. For Model Selection Based on Output (MSBO), we train an ensemble of L models once a new distribution is discovered. The training takes 5 hours on a single GPU. We use Mask R-CNN to annotate the training samples since the ensemble model consists of image classifiers. Mask R-CNN takes approximately 30 minutes to generate labels on a single GPU. In the following Sections, we present a series of experiments to quantify the performance of the DI, MSBO, and MSBI algorithms and compare them against ODIN [64]. Additionally, we evaluate the end-to-end performance of our system against ODIN as well as two state-of-the-art object detectors; YOLOv7 [46] and Mask R-CNN [25].

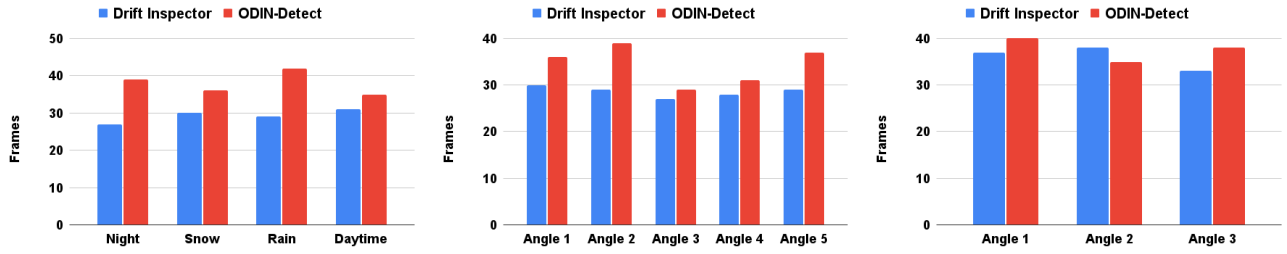
We briefly review ODIN below; full details are available elsewhere [64]. While processing a video stream, ODIN-Detect maintains a collection of clusters, each encompassing a collection of frames. Additionally, each cluster maintains an upper and a lower distance from the centroid, defining a density band. A band encloses a fraction $\Delta = 0.5$ [65] of the frames in the cluster. When a frame is assigned to a cluster, ODIN-Detect measures the distance between the frame and the centroid. When it falls between the upper/lower boundaries, the frame is also assigned to the cluster density band, and the boundaries are updated. If a frame cannot be assigned to an existing cluster, ODIN-Detect creates a temporary cluster for the frame. When the KL divergence of the temporary cluster, before and after adding a frame, is close to zero (specifically 0.007 [65]), the cluster becomes permanent and the ODIN-Specialize trains a model for this cluster. As frames arrive, ODIN assigns new observations, to one or more permanent clusters based on the frame distance from the cluster centroid. If the frame falls to a single cluster, the model associated with the cluster is used. Alternatively, ODIN creates an ensemble using the models of each cluster, assigning equal weight to all participating models. We use the implementation of ODIN as available online [65] in our evaluation.

6.1 Data Drift Detection Evaluation

In this Section, we present the evaluation of the *Drift Inspector* (DI) algorithm of Section 4 and compare it to ODIN-Detect. Note that Algorithm DI proclaims a drift when the distribution of the stream changes to either a known distribution (one for which we have already available models at hand) or an unknown distribution (one that we have not encountered before and for which no model is available). Conversely, ODIN-Detect declares a drift only when a novel distribution appears in the stream and follows a different processing strategy when frames from a previously seen distribution are encountered. Thus, we utilize video frame sequences of unknown distributions to compare all applicable approaches. In the following experiments, we assess the precise frame that each algorithm declares a drift, compared to ground truth as well as the overall time required by each algorithm to detect the drift. Figure 3 presents the precise frame in which DI and ODIN-Detect declare the drift (ground truth drift takes place at frame 0) for the BDD, Detrac, and Tokyo datasets. For DI, we set W=3 and the significance level $r = 0.5$ as empirically we determined that even a very small W is sufficient to attain excellent accuracy. The algorithm demonstrates extremely low dependency on the value of W. We set K=5, which determines the number of nearest neighbors in the training set. The algorithm performance shows nominal dependency on the value of K, as the training data originate from the same distribution, resulting in similar distances to incoming frames. We optimize parameters in ODIN-Detect to minimize both the processing time before detecting a drift and the number of false positive drift detections.

6.1.1 Drift Detection performance. Figure 3(a) presents the results for the **BDD** dataset. BDD contains sequences under different times of day and weather conditions. We observe that algorithm DI processes consistently fewer frames than ODIN-Detect before it declares a drift. On average, across the different sequences of frames, DI declares drift after processing 28 frames, versus 38 required by ODIN-Detect. Figures 3(b) and 3(c), present the corresponding results on **Detrac** and **Tokyo** datasets, where a drift occurs when the camera angle changes. The results are consistent; on average, for **Detrac** and **Tokyo**, DI requires 29 frames in comparison to 36 frames of ODIN-Detect. While DI outperforms ODIN-Detect in BDD and Detrac datasets, ODIN-Detect is detecting a drift faster in Tokyo dataset, on Angle 2. In the Tokyo dataset, Angles 1 and 3 share part of their field of view, while Angle 2 does not. Since ODIN-Detect uses clustering for detecting data drift, the clusters created for Angles 1 and 3 are much closer together compared to the cluster for Angle 2. As such, ODIN-Detect detects drift faster in this case. The results above confirm that DI is on par or slightly better than ODIN-Detect in terms of how many frames need to be processed to identify the drift. Its advantages are evident when the processing time to detect the drift is considered.

6.1.2 Time performance. Table 6 presents the required time to process incoming frames and detect a drift, for DI and ODIN-Detect [64] on 3 datasets. As it is evident, the performance advantages of DI are extremely large. We observe DI requires at least 50% less time than ODIN-Detect to process a video stream while monitoring for potential data drifts. Specifically, ODIN-Detect requires around 6 ms per frame to execute. The execution of the VAE requires around 1 ms while the process to estimate the centroid and the delta bands of the temporary cluster requires on average 4 ms. Finally, ODIN-Detect measures the KL-divergence between the state of the cluster before and after adding a new frame to assess the data drift. In contrast, DI algorithm requires around 3 ms per frame as well as 1 ms to process the frame using the VAE.



(a) BDD data set (b) Detrac data set (c) Tokyo data set

Figure 3: Data drift detection on BDD, Detrac, and Tokyo data sets

6.1.3 Slow Drift Setting. In this section, we compare the performance of the DI algorithm against ODIN-Detect on a slow drift setting, by gathering frames from a live camera in Tokyo for a single day. Additionally, we captured frames from a previous day to train two VAE; one for day frames and one for frames captured during the night. We determine the point where the distribution is changing, based on the actual time of sunrise/sundown in Tokyo for the specific date the video stream was captured. For ODIN-Detect, we conducted extensive hyperparameter tuning for this experiment.

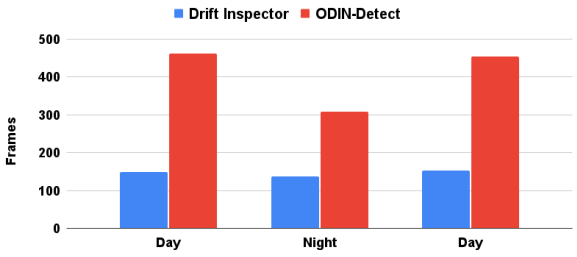


Figure 4: Data drift detection on slow drift setting

Figure 4 demonstrates that DI detects a drift with 3 times fewer processed frames than ODIN-Detect, on average, during a natural transition to a different distribution. ODIN-Detect employs a clustering-based algorithm to determine whether a drift has occurred. Frames captured post-drift remain visually similar to those pre-drift. Thus, ODIN-Detect assigns them to the cluster created for pre-drift frames. The time performance of DI and ODIN-Detect matches the analysis in Table 6. Model selection analysis is omitted, as it parallels the evaluation in Section 6.2.

6.2 Model Selection Evaluation

We present an evaluation of our model selection algorithms MSBI and MSBO of Section 5 and compare them to ODIN-Select [64]. We use the **BDD**, **Detrac**, and **Tokyo** datasets as before. Figure 6 shows the results for all datasets. Figure 6(a) consists of 4 sequences, where each sequence contains 20K frames (corresponding to different weather conditions), and Figure 6(b) and Figure 6(c) contains 5 (6K frames per sequence) and 3 sequences (15K frames per sequence), respectively, each captured by a different camera angle.

MSBO uses a window of frames W_T from the new distribution to evaluate the predictive uncertainty of the models. We experimentally determine that even a very small set of frames is enough to attain superior accuracy for the algorithm; we use W_T=10 in our experiments. MSBI uses W_N=10 frames to evaluate upon, the window of observations W=3 for the martingales, the significance level r=0.5. We set the number of nearest neighbors K=5, to measure the non-conformity score of an input observation.

To compare the various approaches, we measure the model invocations each algorithm incurs. Notice that for MBSO and MBSI, only one model is selected to process frames coming after a drift. ODIN-Select processes each new observation on demand by assigning the observation to available clusters. As such, it can assign a single model to process the frame, or an ensemble of models if the frame has been assigned to more than one cluster. In the *Night* sequence of the BDD dataset, ODIN-Select identifies the night model for 96.3% of the frames while for the rest of the frames generates an ensemble that consists of models *Night* and *Day* with equal weights [(*Night*, 0.5), (*Day*, 0.5)]. As such, for the remaining 3.7% of the frames, two models are used. When a frame cannot be assigned to any of the permanent clusters, the frame is added to a temporary cluster which may subsequently become permanent [65].

6.2.1 Model Selection performance. As we observe in Figure 6, MSBO and MSBI outperform ODIN-Select in terms of the actual number of model inferences required to process the stream. ODIN-Select uses clusters to drive the model selection process, driven by the proximity of incoming frames to cluster centroids. When the input frame is substantially different from all clusters except one (e.g., as in the case of *Day*/*Night* clusters), the number of models involved in processing a frame is comparable to that of MSBO and MSBI. In contrast, when a frame is close to many clusters substantially more models are used. For example, for the *Rain* sequence, ODIN-Select processes frames belonging to the *Rain* sequence using the *Night* model. Both MSBO and MSBI process the video streams with a single model inference per frame as depicted in Figure 6.

6.2.2 Time performance. Table 8 presents a time performance comparison between MSBO, MSBI, and ODIN-Select. We measure the time the algorithm requires to select a suitable model, from a collection of available models, when a drift is detected. Specifically, we observe that MSBO and MSBI algorithms outperform ODIN-Select by one order of magnitude for the time required to select a new model. ODIN-Select assigns each incoming observation to a cluster based on the distance of the observation from the cluster centroid.

For the Detrac dataset, ODIN-Select requires 17.8 ms per frame, as we observe in Table 7. First, ODIN-Select considers all clusters, and if a frame distance from the cluster centroid is within its

Table 7: Per frame model selection time performance

Algorithms	MSBO	MSBI	ODIN-Select
Time (ms)	830	640	17.8

Table 8: Model selection time performance (in seconds)

Dataset	Availab. Models	MSBO	MSBI	ODIN-Select
BDD	4	5.015	22.36	764.4
Detrac	5	8.34	19.57	446.8
Tokyo	3	4.63	13.44	656.1

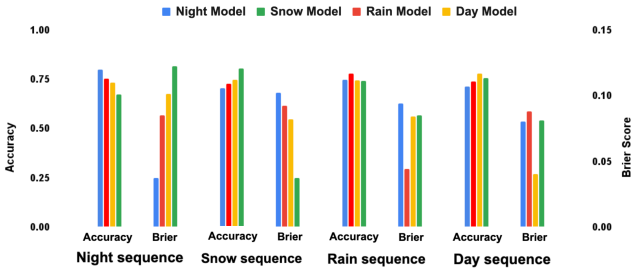


Figure 5: Brier score/Accuracy comparison on BDD dataset

delta band the model associated with the cluster is selected to process the frame. This operation requires for each cluster 3.2 ms (including the delta band boundaries adjustment, the computation of the frame's distance from the cluster's centroid, and the frame assignment to one or more clusters). Since Detrac has 5 clusters, the total is 16 ms. An additional 1.8 ms is required to process the frame using the VAE. MSBO requires 8.3 seconds to select a suitable model after a drift on the Detrac dataset, by inspecting only 10 frames, while MSBI requires 19.6 seconds, by examining 30 frames. Thus, MSBO requires 830 ms per frame, while MSBI requires 640 ms.

As observed, ODIN-Select outperforms MSBO and MSBI in per-frame processing cost. However, ODIN-Select performs model selection on each incoming frame, while MSBO and MSBI process only a subset of frames before selecting a suitable model. Our proposals result in one order of magnitude faster model selection, compared to ODIN-Select for real video streams with thousands of frames. To further showcase our proposals superiority, we modified ODIN-Select to select a model within a frame window, similar to MSBO and MSBI. The modified ODIN-Select exhibits better model selection time performance than MSBO and MSBI when compared in isolation. However, the modified version significantly hurts the model invocations, end-to-end time performance, and query accuracy.

6.2.3 Accuracy vs Brier Score. Since MSBO achieves better model selection accuracy compared to ODIN-Select, we conducted one more experiment to highlight the advantages of Brier score versus classification accuracy. Due to space constraints, we present the experiment for the **BDD** dataset. In this experiment, we use the BDD dataset, thus, we trained 4 models for Night, Day, Rain, and Snow sequences. In Figure 5, we depict the Brier score and the image classification accuracy of each model for each sequence of the BDD dataset. For each sequence, we use the model associated with the sequence, for example, the *Night* model is deployed when we process the Night sequence. The left y-axis depicts the accuracy while the right y-axis the Brier score. We observe that the classification accuracy in all cases differs only by 10% from the model with the highest accuracy. Conversely, the Brier score of the most accurate model is 2 times lower compared to the rest of the models. Selecting a model based on accuracy may lead to wrong selections due to potential classification noise (differences in accuracy are marginally different). The Brier score provides a better separation of the resulting values and yields more robust decisions.

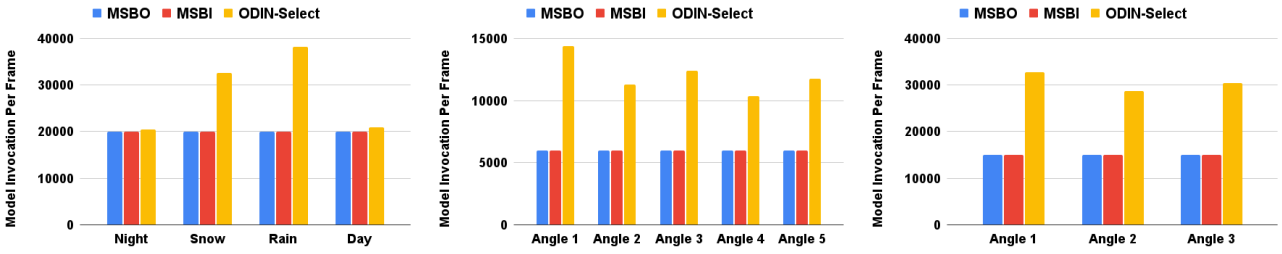
6.3 End-to-End Performance Evaluation

In this Section, we present the end-to-end performance evaluation of our proposals. In particular, we benchmark the algorithms DI (drift detection) utilizing MSBO (DI, MSBO) and MSBI (DI, MSBI) (model selection). For ODIN, we use ODIN-Detect (drift detection), and ODIN-Select (model selection). We also compare against two state-of-the-art object detection frameworks oblivious to drift, namely, YOLOv7 [46] and Mask R-CNN [25]. We perform experiments measuring the end-to-end time required to detect a drift and select a model. To evaluate the practical significance of a superior drift detection and model selection framework, we measure the accuracy of count and spatial constrained queries over a video stream embodying diverse drifts. The count query measures the number of cars appearing in the video stream for each frame, while the spatial-constrained query determines the position of cars and buses in the video stream. If the drift detection and model selection framework do not perform well, query accuracy will be compromised.

6.3.1 Count Query. For the count query, we use VGG-19 [62] to train classifiers for all the datasets as per Table 8. For the BDD dataset, we train 4 models, namely, *Night*, *Day*, *Rain*, and *Snow*, on data captured from the respective sequences (i.e. a Night model is trained on frames captured during the night). We follow the same approach for Detrac and Tokyo datasets. We use the same classifiers for (DI, MSBO), (DI, MSBI), and (ODIN-Detect, ODIN-Select). When a drift is declared, MSBI, MSBO, and ODIN-Select, choose a model (or an ensemble of models for ODIN-Select) to process frames after the drift. Then, the selected models are deployed. For YOLOv7 and Mask R-CNN, we perform object detections for all frames of the dataset. Finally, we use Mask R-CNN to generate labels that consist of the number of cars appearing in each frame.

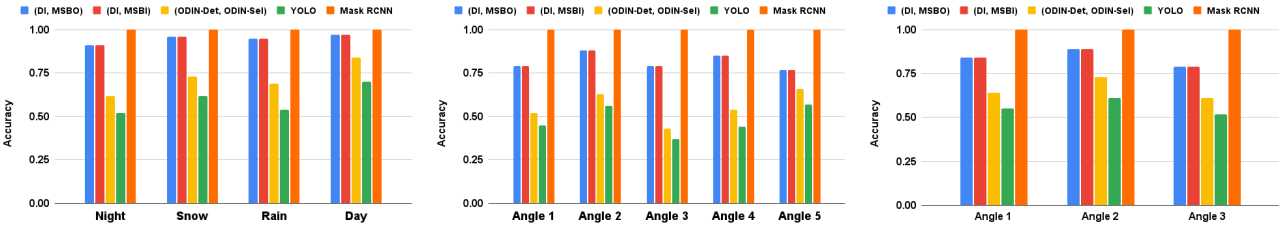
Table 9 presents the end-to-end time performance of (DI, MSBO), (DI, MSBI), and (ODIN-Detect, ODIN-Select); we measure the time required to detect a drift and the time to select a suitable model after the drift. In Table 9 we also present the time YOLOv7 and Mask R-CNN take to perform object detections on all frames of the dataset. We observe that the time required by (DI, MSBO) is much smaller than (ODIN-Detect, ODIN-Select) and slightly faster than (DI, MSBI). (DI, MSBO) require 278.4 seconds to process the entire **BDD** dataset, while (ODIN-Detect, ODIN-Select) require 1400.6 seconds, 3 times more than (DI, MSBO). Similarly (DI, MSBO) achieve the best time for **Detrac** and **Tokyo** datasets. ODIN requires 4-6 times more time for the same task, similar to YOLOv7, while Mask R-CNN is one order of magnitude slower than our proposals.

Next, we investigate the query accuracy A_q of count queries on all datasets. The accuracy A_q measures the fraction of the frames where the classifier prediction matches the ground truth. Specifically, we predict the number of cars appearing in a frame, where the ground truth consists of the true number of cars in



(a) BDD data set (b) Detrac data set (c) Tokyo data set

Figure 6: Model invocation per frame on BDD, Detrac, and Tokyo data sets



(a) BDD data set (b) Detrac data set (c) Tokyo data set

Figure 7: Count query accuracy on BDD, Detrac, and Tokyo data set

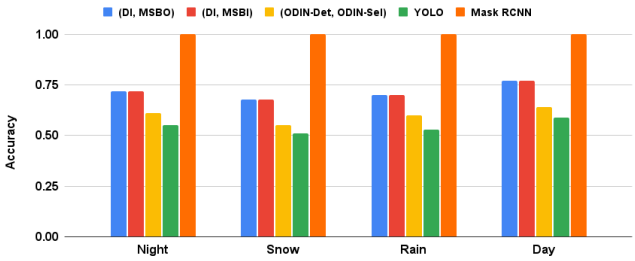


Figure 8: Spatial constrained query accuracy on BDD dataset

each frame. Figure 7 presents A_q on all datasets. We observe that (DI, MSBO), and (DI, MSBI) outperform (ODIN-Detect, ODIN-Select) by 40% on query accuracy in all cases, and 50% compared to YOLOv7. Mask R-CNN achieves perfect accuracy because it was used as the baseline to annotate the datasets. The superior model selection capabilities of Algorithm MSBO lead to higher A_q compared to ODIN since MSBO always deploys the most accurate model to process incoming frames.

6.3.2 Spatial Constrained Query. For the spatial constrained query, we use the OD-CLF filter [37], to train models for all datasets as per Table 8. We follow the same approach as with the count query. The same models are used to evaluate (DI, MSBO), (DI, MSBI), and (ODIN-Detect, ODIN-Select). The time performance of this query is the same as for the count query (Table 9). We use Mask R-CNN to extract the positions of cars and buses appearing in the video frames. Query accuracy A_q measures the fraction of frames where the OD-CLF filter prediction matches the ground truth. The query predicate is formed as "bus is on the left side of a car", while the ground truth consists of frames that match the condition. Due to space constraints, we only show the results of the BDD dataset, in Figure 8. The results are consistent with the count query; (DI, MSBO) outperforms (ODIN-Detect, ODIN-Select) in all sequences by achieving 20% higher A_q, while it is 3 times faster.

Table 9: End-to-end performance results (in seconds)

Dataset	(DI, MSBO)	(DI, MSBI)	(ODIN-Detect, ODIN-Select)	YOLO	Mask RCNN
BDD	278.4	295.8	1400.6	1231	10680
Detrac	105.6	116.8	682.6	462	4005
Tokyo	169.2	178	950.1	692	6007.5

7 CONCLUSIONS

We considered the problem of detecting and mitigating data drifts occurring during the processing of real video streams. We presented the *Drift Inspector* along with MSBO and MSBI algorithms and demonstrated that together constitute a robust set of approaches for dealing with data drift in video streams. Our experimental results demonstrated vast performance advantages compared with state-of-the-art proposals while attaining superior accuracy.

Our work raises avenues for further work in this area. First, incorporating our proposals in a video analytics system is already underway. Second, although we have demonstrated the efficacy of conformal martingales, further exploration and tuning will provide further insight on how much we can push the boundary of efficiency in video stream change detection. Finally, the state of the art in video stream querying is under rapid evolution, incorporating other semantics such as object interactions [75]. It would be interesting to explore change detection in the presence of activity querying.

REFERENCES

[1] Romil Bhardwaj , Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Victor Bahl, and Ion Stoica. 2022. Ekya: Continuous Learning of Video Analytics Models on Edge Compute Servers. In *USENIX NSDI*. https://www.microsoft.com/en-us/research/publication/ekya-continuous-learning-of-video-analytics-models-on-edge-compute-servers/

[2] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U. Rajendra Acharya, and et al. 2021. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion* 76 (Dec 2021), 243–297. https://doi.org/10.1016/j.inffus.2021.05.008

[3] Michèle Basseville and Igor Nikiforov. 1993. *Detection of Abrupt Change Theory and Application*. Vol. 15.

[4] Favyen Bastani, Songtao He, Arjun Balasingam, Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan, Michael Cafarella, Tim Kraska, and Sam Madden. 2020. MIRIS: Fast Object Track Queries in Video. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 1907–1921. https://doi.org/10.1145/3318464.3389692

[5] Favyen Bastani and Samuel Madden. 2022. OTIF: Efficient tracker pre-processing over large video datasets. In *Proceedings of the 2022 International Conference on Management of Data*. 2091–2104.

[6] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin A. Zinkevich. 2019. Data Validation for Machine Learning. In *Conference on Machine Learning and Systems*. https://api.semanticscholar.org/CorpusID:182180482

[7] Leo Breiman. 1996. Bagging Predictors. *Mach. Learn.* 24, 2 (Aug. 1996), 123–140. https://doi.org/10.1023/A:1018054314350

[8] G. Brier. 1950. VERIFICATION OF FORECASTS EXPRESSED IN TERMS OF PROBABILITY. *Monthly Weather Review* 78 (1950), 1–3.

[9] B. Brodsky and B. Darkhovsky. 1993. Nonparametric Methods in Change Point Problems.

[10] Daren Chao, Kaiwen Chen, and Nick Koudas. 2023. SVQ-ACT: Querying for Actions over Videos. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 3599–3602.

[11] Daren Chao, Yueting Chen, Nick Koudas, and Xiaohui Yu. 2023. Track merging for effective video query processing. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 164–176.

[12] Daren Chao, Nick Koudas, and Ioannis Xarchakos. 2020. SVQ++: Querying for Object Interactions in Video Streams. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 2769–2772. https://doi.org/10.1145/3318464.3384701

[13] Daren Chao, Nick Koudas, and Xiaohui Yu. 2023. Marshalling Model Inference in Video Streams. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. 1379–1392. https://doi.org/10.1109/ICDE55515.2023.00110

[14] Yueting Chen, Xiaohui Yu, and Nick Koudas. 2020. TQVS: Temporal Queries over Video Streams in Action. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2737–2740.

[15] Yueting Chen, Xiaohui Yu, and Nick Koudas. 2021. Evaluating Temporal Queries Over Video Feeds. *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data* (2021).

[16] Yueting Chen, Xiaohui Yu, and Nick Koudas. 2022. Ranked Window Query Retrieval over Video Repositories. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2776–2791.

[17] Bradley Efron and Robert J Tibshirani. 1994. *An Introduction to the Bootstrap*. CRC press.

[18] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. arXiv:stat.ML/1506.02142

[19] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, Muhammad Shahzad, Wen Yang, Richard Bamler, and Xiao Xiang Zhu. 2021. A Survey of Uncertainty in Deep Neural Networks. arXiv:cs.LG/2107.03342

[20] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. 580–587. https://doi.org/10.1109/CVPR.2014.81

[21] Tilmann Gneiting and Adrian E Raftery. 2007. Strictly Proper Scoring Rules, Prediction, and Estimation. *J. Amer. Statist. Assoc.* 102, 477 (2007), 359–378. https://doi.org/10.1198/016214506000001437

[22] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On Calibration of Modern Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017 (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, 1321–1330. http://proceedings.mlr.press/v70/guo17a.html

[23] Brandon Haynes, Maureen Daum, Dong He, Amrita Mazumdar, Magdalena Balazinska, Alvin Cheung, and Luis Ceze. 2021. VSS: A Storage System for Video Analytics. In *Proceedings of the 2021 International Conference on Management of Data*. 685–696.

[24] Brandon Haynes, Maureen Daum, Amrita Mazumdar, Magdalena Balazinska, Alvin Cheung, and Luis Ceze. 2020. VisualWorldDB: A DBMS for the Visual World. In *CIDR*.

[25] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2018. Mask R-CNN. arXiv:cs.CV/1703.06870

[26] Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. 2019. Why ReLU Networks Yield High-Confidence Predictions Far Away From the Training Data and How to Mitigate the Problem. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019*. Computer Vision Foundation / IEEE, 41–50. https://doi.org/10.1109/CVPR.2019.00013

[27] Tom Heskes. 1996. Practical Confidence and Prediction Intervals. In *Proceedings of the 9th International Conference on Neural Information Processing Systems (NIPS'96)*. MIT Press, Cambridge, MA, USA, 176–182.

[28] Shen-Shyang Ho. 2005. A Martingale Framework for Concept Change Detection in Time-Varying Data Streams. In *Proceedings of the 22nd International Conference on Machine Learning (ICML '05)*. Association for Computing Machinery, 321–327. https://doi.org/10.1145/1102351.1102392

[29] 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 *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, 269–286. https://www.usenix.org/conference/osdi18/presentation/hsieh

[30] Roberto Jordaney, Kumar Sharad, Santanu K. Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting Concept Drift in Malware Classification Models. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 625–642. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jordaney

[31] Frank J. Massey Jr. 1951. The Kolmogorov-Smirnov Test for Goodness of Fit. *J. Amer. Statist. Assoc.* 46, 253 (1951), 68–78. https://doi.org/10.1080/01621459.1951.10500769 arXiv:https://www.tandfonline.com/doi/pdf/10.1080/01621459.1951.10500769

[32] D. Kang, P. Bailis, and M. Zaharia. 2019. BlazeIT: Fast Exploratory Video Queries Using Neural Networks. In *PVLDB*.

[33] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Neural Network Queries over Video at Scale. *Proc. VLDB Endow.* 10, 11 (Aug. 2017), 1586–1597. https://doi.org/10.14778/3137628.3137664

[34] Alex Kantchelian, Sadia Afroz, Ling Huang, Aylin Caliskan, Brad Miller, Michael Carl Tschantz, Rachel Greenstadt, Anthony D. Joseph, and J. Doug Tygar. 2013. Approaches to adversarial drift. *Proceedings of the 2013 ACM workshop on Artificial intelligence and security* (2013). https://api.semanticscholar.org/CorpusID:21857

[35] Diederik P. Kingma and M. Welling. 2014. Auto-Encoding Variational Bayes. *CoRR* abs/1312.6114 (2014).

[36] Anoop Korattikara, Vivek Rathod, Kevin Murphy, and Max Welling. 2015. Bayesian Dark Knowledge. arXiv:cs.LG/1506.04416

[37] Nick Koudas, Raymond Li, and Ioannis Xarchakos. 2020. Video Monitoring Queries. In *Proceedings of IEEE ICDE*.

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* 60, 6 (May 2017), 84–90. https://doi.org/10.1145/3065386

[39] Ziliang Lai, Chenxia Han, Chris Liu, Pengfei Zhang, Eric Lo, and Ben Kao. 2021. Top-K Deep Video Analytics: A Probabilistic Approach. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)*. Association for Computing Machinery, 1037–1050. https://doi.org/10.1145/3448016.3452786

[40] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. arXiv:stat.ML/1612.01474

[41] Zhizhong Li and Derek Hoiem. 2020. Improving Confidence Estimates for Unfamiliar Examples. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*. IEEE, 2683–2692. https://doi.org/10.1109/CVPR42600.2020.00276

[42] Viktor Losing, Barbara Hammer, and Heiko Wersing. 2016. KNN Classifier with Self Adjusting Memory for Heterogeneous Concept Drift. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 291–300. https://doi.org/10.1109/ICDM.2016.0040

[43] Christos Louizos and Max Welling. 2016. Structured and Efficient Variational Deep Learning with Matrix Gaussian Posteriors. arXiv:stat.ML/1603.04733

[44] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan Zhang. 2019. Learning under Concept Drift: A Review. *IEEE Transactions on Knowledge and Data Engineering* 31, 12 (2019), 2346–2363. https://doi.org/10.1109/TKDE.2018.2876857

[45] 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 (SIGMOD '18)*. ACM, 1493–1508. https://doi.org/10.1145/3183713.3183751

[46] Jiayi Ma, Xingyu Jiang, Aoxiang Fan, Junjun Jiang, and Junchi Yan. 2021. Image Matching from Handcrafted to Deep Features: A Survey. *Int. J. Comput. Vision* 129, 1 (jan 2021), 23–79. https://doi.org/10.1007/s11263-020-01359-2

[47] Federico Maggi, William Robertson, Christopher Kruegel, and Giovanni Vigna. 2009. Protecting a Moving Target: Addressing Web Application Concept Drift, Vol. 5758. 21–40. https://doi.org/10.1007/978-3-642-04342-0_2

[48] Oscar Moll, Favyen Bastani, Sam Madden, Mike Stonebraker, Vijay Gadepally, and Tim Kraska. 2022. Exsample: Efficient searches on video repositories through adaptive sampling. In *2022 IEEE 38th International Conference on Data*

Engineering (ICDE). IEEE, 2956–2968.

[49] George V. Moustakides. 1986. Optimal Stopping Times for Detecting Changes in Distributions. *The Annals of Statistics* 14, 4 (1986), 1379 – 1387. https://doi.org/10.1214/aos/1176350164

[50] Pravin Nagar, Mansi Khemka, and Chetan Arora. 2020. Concept Drift Detection for Multivariate Data Streams and Temporal Segmentation of Daylong Egocentric Videos. In *Proceedings of the 28th ACM International Conference on Multimedia*. Association for Computing Machinery, 1065–1074. https://doi.org/10.1145/3394171.3413713

[51] Radford M Neal. 2012. *Bayesian learning for neural networks*. Vol. 118. Springer Science & Business Media.

[52] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D Sculley, Sebastian Nowozin, Joshua V. Dillon, Balaji Lakshminarayanan, and Jasper Snoek. 2019. Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty Under Dataset Shift. arXiv:stat.ML/1906.02530

[53] E. S. PAGE. 1954. CONTINUOUS INSPECTION SCHEMES. *Biometrika* 41, 1-2 (06 1954), 100–115. https://doi.org/10.1093/biomet/41.1-2.100

[54] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

[55] Karl Pearson. 1893. Asymmetrical frequency curves. *Nature* 48, 1252 (1893), 615–616.

[56] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence. 2009. *Dataset Shift in Machine Learning*. The MIT Press.

[57] Stephan Rabanser, Stephan Günnemann, and Zachary Chase Lipton. 2018. Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift. In *Neural Information Processing Systems*. https://api.semanticscholar.org/CorpusID:53096511

[58] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. 2014. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *Proceedings of the 31st International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Vol. 32. PMLR, 1278–1286. https://proceedings.mlr.press/v32/rezende14.html

[59] Francisco Romero, Caleb Winston, Johann Hauswald, Matei A. Zaharia, and Christos Kozyrakis. 2023. Zelda: Video Analytics using Vision-Language Models. *ArXiv* abs/2305.03785 (2023). https://api.semanticscholar.org/CorpusID:258557909

[60] A. N. Shiryaev. 1963. On Optimum Methods in Quickest Detection Problems. *Theory of Probability & Its Applications* 8, 1 (1963), 22–46. https://doi.org/10.1137/1108002 arXiv:https://doi.org/10.1137/1108002

[61] Connor Shorten and Taghi M. Khoshgoftaar. 2019. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data* 6 (2019), 1–48.

[62] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv 1409.1556* (09 2014).

[63] Eduardo Spinosa, Andre de Carvalho, and João Gama. 2007. OLINDDA: a cluster-based approach for detecting novelty and concept drift in data streams. 448–452.

[64] Abhijit Suprem, Joy Arulraj, Calton Pu, and Joao Ferreira. 2020. ODIN: Automated Drift Detection and Recovery in Video Analytics. *Proc. VLDB Endow.* 13, 12 (July 2020), 2453–2465. https://doi.org/10.14778/3407790.3407837

[65] Abhijit Suprem, Joy Arulraj, Calton Pu, and Joao Ferreira. 2020. ODIN: Automated Drift Detection and Recovery in Video Analytics. https://github.com/asuprem/ODIN.

[66] Da Tang, Dawen Liang, Tony Jebara, and Nicholas Ruozzi. 2019. Correlated Variational Auto-Encoders. *CoRR* abs/1905.05335 (2019). arXiv:1905.05335 http://arxiv.org/abs/1905.05335

[67] Kurt Thomas, Chris Grier, Justin Ma, Vern Paxson, and Dawn Song. 2011. Design and Evaluation of a Real-Time URL Spam Filtering Service. *Proceedings - IEEE Symposium on Security and Privacy*, 447–462. https://doi.org/10.1109/SP.2011.25

[68] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2020. Ensemble Adversarial Training: Attacks and Defenses. arXiv:stat.ML/1705.07204

[69] Denis Volkhonskiy, Ilia Nouretdinov, Alexander Gammerman, Vladimir Vovk, and Evgeny Burnaev. 2017. Inductive Conformal Martingales for Change-Point Detection. *Proceedings of Machine Learning Research* 60 (06 2017), 132–153.

[70] Vladimir Vovk, Alex Gammerman, and Glenn Shafer. 2005. *Algorithmic Learning in a Random World*. Springer-Verlag, Berlin, Heidelberg.

[71] Vladimir Vovk, Ilia Nouretdinov, and Alexander Gammerman. 2003. Testing Exchangeability On-Line. 768–775.

[72] Longyin Wen, Dawei Du, Zhaowei Cai, Zhen Lei, Ming-Ching Chang, Honggang Qi, Jongwoo Lim, Ming-Hsuan Yang, and Siwei Lyu. 2015. DETRAC: A New Benchmark and Protocol for Multi-Object Tracking. *CoRR* abs/1511.04136 (2015). arXiv:1511.04136 http://arxiv.org/abs/1511.04136

[73] Yeming Wen, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger Grosse. 2018. Flipout: Efficient Pseudo-Independent Weight Perturbations on Mini-Batches. arXiv:cs.LG/1803.04386

[74] Ioannis Xarchakos and Nick Koudas. 2019. SVQ: Streaming Video Queries. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19)*. Association for Computing Machinery, 2013–2016. https://doi.org/10.1145/3299869.3320230

[75] Yannis Xarchakos and Nick Koudas. 2021. Querying for Interactions. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 2153–2158. https://doi.org/10.1109/ICDE51399.2021.00216

[76] Zhuangdi Xu, Gaurav Tarlok Kakkar, Joy Arulraj, and Umakishore Ramachandran. 2022. EVA: A Symbolic Approach to Accelerating Exploratory Video Analytics with Materialized Views. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 602–616. https://doi.org/10.1145/3514221.3526142

[77] Piyush Yadav, Dhaval Salwala, Felipe Arruda Pontes, Praneet Dhingra, and Edward Curry. 2021. Query-Driven Video Event Processing for the Internet of Multimedia Things. *Proc. VLDB Endow.* 14, 12 (jul 2021), 2847–2850. https://doi.org/10.14778/3476311.3476360

[78] 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. arXiv:cs.CV/1805.04687