MATCHMAKER: DATA DRIFT MITIGATION IN MACHINE LEARNING FOR LARGE-SCALE SYSTEMS

Ankur Mallick [1] **Kevin Hsieh** [2] **Behnaz Arzani** [2] **Gauri Joshi** [1]

ABSTRACT

Today's data centers rely more heavily on machine learning (ML) in their deployed systems. However, these systems are vulnerable to the *data drift* problem, that is, a mismatch between training and test data, which can lead to significant performance degradation and system inefficiencies. In this paper, we demonstrate the impact of data drift in production by studying two real-world deployments in a leading cloud provider. Our study shows that, despite frequent model retraining, these deployed models experience major accuracy drops (up to 40%) and high accuracy variation, which lead to drastic increase in operational costs. None of the current solutions to the data drift problem are designed for large-scale deployments, which need to address real-world issues such as scale, ground truth latency, and mixed types of data drift. We propose Matchmaker, the first scalable, adaptive, and flexible solution to the data drift problem in large-scale production systems. Matchmaker finds the most *similar* training data batch and uses the corresponding ML model for inference on *each* test point. As part of Matchmaker we introduce a novel similarity metric to address multiple types of data drifts while only incurring limited overhead. Experiments on our two real-world ML deployments show Matchmaker significantly improve model accuracy (upto 14% and 2%), which saves 18% and 1% in the operational costs. At the same time, Matchmaker provides 8× and 4× faster predictions than a state-of-the-art ML data drift solution, AUE.

1 INTRODUCTION

Machine learning (ML) is ubiquitous in making intelligent decisions for data centers, which need to meet ever-increasing demands while achieving various systems objectives such as performance, cost efficiency, and reliability. Operators have deployed ML models for various tasks such as resource management (e.g., (Cortez et al., 2017; Balaji et al., 2020; Rzadca et al., 2020)), power management (e.g., (Gao, 2014; Hossain et al., 2017)), data indexing (e.g., (Abu-Libdeh et al., 2020)), temperature management (e.g, (Lazic et al., 2018)), and failure/compromise detection (e.g., (Zhang et al., 2017; Arzani et al., 2020)).

The accuracy of these ML models often has significant impact on the performance, security, and efficiency of data center operations (Song et al., 2020; Flynn et al., 2019; Cortez et al., 2017; Gao et al., 2020; Arzani et al., 2020). Drops in accuracy can lead to consequences such as system performance degradation and cost increase. Our experience in a large cloud provider shows that, despite frequent retraining, many of the deployed ML models suffer from frequent and significant (up to 40%) accuracy drops which, in turn, can result in drastic increase in operational costs. These drops are typically due to *data drift*.

Unlike ML applications such as image classification (LeCun et al., 1989; Russakovsky et al., 2015) and natural language processing (Chowdhury, 2003) where the data typically comes from a *stable* underlying sample space, most systems applications involve data that is inherently temporal and can change over time. These changes occur because real-world systems constantly face internal and external changes such as system upgrades, configuration changes, new workloads, and surging user demands (Sculley et al., 2014). This results in the data drift problem: a mismatch between the training and test data. Data drift leads to significant accuracy drops because the assumption on which most ML models are trained, that the training data in the *past* is sufficiently similar to the test data in the *future*, is violated.

The ML community classifies the data drift problem into two broad categories (Moreno-Torres et al., 2012; Gama et al., 2014): (1) *covariate shift* (Shimodaira, 2000) or virtual drift, where training data distribution changes over time but the underlying concept (the mapping from features to labels) stays the same; and (2) *concept drift* (Widmer & Kubat, 1996) or real drift, where the underlying concept changes over time. ML practitioners today manage this problem by (a) periodic *model retraining* to incorporate recent data into

*Equal contribution [1]Carnegie Mellon University [2]Microsoft. Correspondence to: Ankur Mallick <amallic1@andrew.cmu.edu>, Kevin Hsieh <Kevin.Hsieh@microsoft.com>.

Preliminary work. Under review by the Machine Learning and Systems (MLSys) Conference. Do not distribute.

their ML models; and (b) re-weighing the training data to increase the importance of more recent data during retraining. However, these retraining strategies are static, and they are not designed to handle different types of data drifts. Indeed, we find deployed strategies continue to experience major accuracy drops and variations (see Section 2).

In the ML literature, there are many advanced solutions to tackle covariate shift or concept drift separately (Widmer & Kubat, 1996; Kolter & Maloof, 2007; Bifet & Gavalda, 2007; Elwell & Polikar, 2011; Brzezinski & Stefanowski, 2013; Pesaranghader et al., 2018; Tahmasbi et al., 2020). However, existing data drift solutions have not considered the following practical challenges that arise in deployment:

- **Scalability.** Large-scale systems require running inference on millions of test samples per-day. Most existing data drift solutions do not prioritize scalability and therefore incur a steep inference cost and latency due to this high load of test samples.
- **Delay in obtaining ground truths.** There is usually significant delay in obtaining ground truth in large-scale systems (e.g., several hours to a week): e.g., an operator needs to identify the root cause of an incident before we know which component is at fault. Almost all existing data drift solutions rely on periodic retraining and make many mistakes until they are retrained. The delay in obtaining new labels exacerbates this problem as it limits our ability to retrain these solutions frequently.
- **Mixed data drifts.** Existing solutions are designed for either covariate shift or concept drift, but not both. Real-world data drifts, on the other hand, are unpredictable and can be caused by either of these two problems. Different test points in a single data batch can experience different types of data drifts as well.

We present Matchmaker, the first solution to the data drift problem that is designed for ML deployment in large-scale systems. Matchmaker is a general approach for mitigating data drift that (1) only incurs limited overhead during inference; (2) adapts to data drift without having to wait for a new batch of ground truth labels; and (3) addresses *both* covariate shift and concept drift simultaneously.

Key Techniques. Matchmaker is adaptive, flexible, and runs at scale. Its key underlying idea is to dynamically identify the batch of training data that is most *similar* to *each* test sample, and use the ML model trained on that data for inference. This idea provides three major advantages. First, as Matchmaker selects the matching ML model for each sample at *test time*, it can adapt to data drifts without having to wait for new ground truth labels to retrain. Second, it has the flexibility to make *independent* decisions on each test sample which is more effective than existing data drift solutions that always use the same ML model (or the same set of ML models) for *all* incoming test samples until they do another round of adaptation. Third, Matchmaker provides operators with the means to select their own *similarity* metric: ML practitioners can choose which similarity metric achieves the right trade-off for them in terms of computation overhead and model accuracy. As part of our solution, we present our new similarity metric that measures a combination of covariate shift (i.e., similarity in feature values) and concept drift (i.e., similarity in the mapping from features to labels). This metric allows us to mitigate both covariate shift and concept drift while achieving significantly less computation overheads than existing metrics.

System optimizations. We design Matchmaker for large-scale systems that make millions of predictions every day. Matchmaker uses three key techniques to minimize its overheads during inference: (1) it *splits* the work of computing similarity scores between the offline and online pipelines so that the offline pipeline handles most of the heavy lifting. (2) it trains a *random forest* to create a data partitioning kernel (Davies & Ghahramani, 2014), and derives data similarity based on the data traversal paths. This idea results in a system that is much more scalable than prior art (e.g., those that use Euclidean distance as a similarity metric). Our use of the random forest kernel allows us to only compute the path of each test sample once ($\mathcal{O}(1)$) as opposed to having to compute a pairwise metric across all data points ($\mathcal{O}(N)$). (3) Matchmaker further reduces inference time by using the training samples to create a cache of pre-computed scores. During inference Matchmaker can look up the score for an incoming test sample instead of computing from scratch.

Evaluation Highlights. We evaluate Matchmaker on two real-world applications deployed in a production cloud provider: network incident routing (NETIR) and VM CPU utilization (VMCPU). We evaluate these two applications over a twelve- and three-month period, respectively. Our results show, when compared to the currently deployed retraining mechanisms, Matchmaker (1) improves accuracy, in some instances, by up to 14% for NETIR and up to 2% for VMCPU; and (2) reduces average operational costs by 18% for NETIR and 1% for VMCPU. Compared to a state-of-the-art concept drift solution, AUE (Brzezinski & Stefanowski, 2013), Matchmaker provides $8\times$ and $4\times$ faster ML predictions while achieving comparable accuracy. Compared to a recent concept drift solution DriftSurf (Tahmasbi et al., 2020), Matchmaker provides upto 20% and 6% improvement in accuracy with similar prediction speed. Matchmaker is currently being deployed in NETIR.

Summary of Contributions.

- To our knowledge, this is the first data drift solution designed for ML in large-scale systems that addresses practical issues in real-world deployments such as scale, ground truth latency, and mixed data drifts.

- We introduce a new similarity metric that address both covariate shift and concept drift at the same time. We present an efficient design to compute this metric by splitting the work between an online and offline pipeline.
- We demonstrate the effectiveness of our solution using two real-world, large-scale applications: our solution significantly reduces accuracy variation over deployed retraining strategies, and is significantly faster than state-of-the-art ML data drift solutions.

2 Background and Motivation

We first introduce the background for the data drift problem and then describe two *deployed* ML applications in real-world systems that motivate our solution. We next discuss our observations from these deployments and the design requirements for a data drift solutions for large-scale systems.

2.1 The Data Drift Problem and Its Solutions

The ML community has investigated the data drift problem: the problem of accuracy variation (over time) in ML models. This problem typically occurs because the model runs inference on data (test data) which is significantly different from the data that was used for training it. Several past works have investigated the data drift problem in the context of continuously arriving/stream data and have proposed various solutions (Moreno-Torres et al., 2012; Tahmasbi et al., 2020; Suprem et al.; Bifet & Gavalda, 2007).

In supervised problems (where features $\mathbf{x}$ are used to predict labels y), the major causes of data drift are (Moreno-Torres et al., 2012) - (1) A change in the distribution of features $\mathbf{x}$ (*covariate shift*) e.g., if new kinds of incidents with previously unseen feature values occur, or (2) a change in the underlying relationship between features $\mathbf{x}$, and labels y (*concept drift*) e.g., if a system and its dependencies change such that incidents with certain symptoms are no longer caused by a particular component.

Existing solutions to the data drift problem can be broadly classified as: (a) **window based methods** (e.g., (Widmer & Kubat, 1996)), which use a sliding window over old data to train new models, (b) **shift detection methods** (e.g., (Pesaranghader et al., 2018)), which use statistical tests to detect data drift and retrain models only when they detect that data drift has happened, or (c) **ensemble based methods** (e.g., (Brzezinski & Stefanowski, 2013)), which train an ensemble of models on "old" training data and take a weighted average of their predictions.

2.2 A Case Study of Deployed Applications

Many operators today use ML in deployment for a number of management and security related tasks (Song et al., 2020; Flynn et al., 2019; Cortez et al., 2017; Gao et al., 2020; Arzani et al., 2020). Deployed ML solutions represent a continuous ML paradigm (Baylor et al., 2019) where the ML pipeline must run inference continuously on new and incoming test samples that arrive one at a time. The samples are assigned ground truth labels over time, which could range from hours, to days, or even weeks. Once a batch of samples has been labeled, it can be used as part of the training data to retrain the model. We use $\mathbf{X}_t$ and $\mathbf{y}_t$ to denote the samples and labels in a batch t respectively. Deployed ML pipelines are trained on all or a subset of past batches $(\mathbf{X}_1, \mathbf{y}_1), \ldots, (\mathbf{X}_t, \mathbf{y}_t)$. They will then run inference for new test samples, which after being labeled will form batch $t+1$ and are added to the training data. Here, we study two such deployments from a leading cloud provider[1].

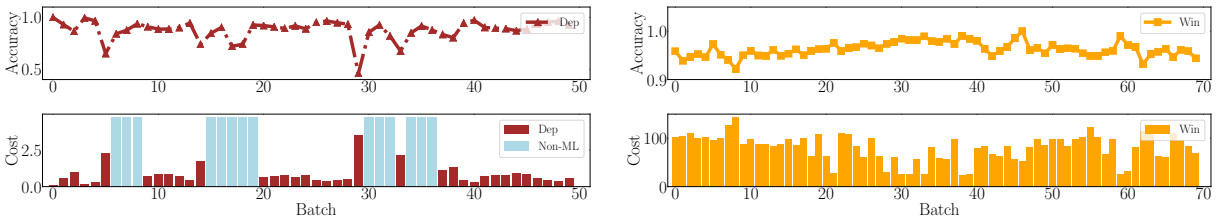
Network Incident Routing (NETIR): Incidents in a data center network include any unintended behavior, reported by customers, operators, or automated monitoring programs (Google), that can adversely impact service availability and performance. Different incidents are handled by different teams, and the ability to route an incident to the right team in the shortest amount of time can significantly reduce the time required to resolve the incident. The operator we study has an ML pipeline to route incidents to the responsible teams. Each point in $\mathbf{X}_t$ is a feature vector containing characteristics of the incident and the labels in $\mathbf{y}_t$ represent whether a particular team is responsible for that incident. The operator trains the model on past (resolved) incidents and retrains the model every week (each batch t contains one week of incidents), while giving more weight to recent incidents. In Fig. 1a we show the normalized accuracy and operational cost of this system over a one year time-frame[2].

VM CPU Utilization (VMCPU): Virtual Machines (VMs) differ in their CPU utilization based on the nature of their workload. Accurately predicting the CPU utilization of incoming VM requests can enable cloud providers to efficiently allocate resources (i.e. allocate more resources to requests with higher CPU utilization). The operator we studied has an ML pipeline categorizing the expected utilization of an incoming VM request using a feature vector in $\mathbf{X}_t$ describing its attributes. The pipeline predicts whether the VM is expected to have high or low CPU utilization and is trained on past requests. The model is retrained every day (each batch t is one day) after it obtains ground truth labels. Fig. 1b shows the normalized accuracy and cost for this system over a three-month period.

We obtain the following insights from our study in these two ML deployments:

[1] We omit citations to these systems to preserve anonymity.

[2] To protect sensitive information all accuracy values are normalized by the maximum accuracy of the currently deployed model over the entire period throughout the paper.



(a) Network Incident Routing

(b) VM CPU Utilization

Figure 1. **(Top)** The deployed models for continuous ML (over a stream of data batches) for Network Incident Routing (50 weeks, one batch/week) and VM CPU Utilization (68 days, one batch/day) suffer from frequent accuracy drops due to to data drifts *despite continuously retraining* at the end of each data batch. **(Bottom)** Normalized operational cost, where the systems sometimes need to revert to Non-ML solutions (blue bars) when the ML solution cannot provide stable accuracy. Even small accuracy drops can incur high costs.

(1) Real-world ML deployments experience significant accuracy drops despite frequent retraining. The top of Fig. 1 shows sharp drops in accuracy in multiple instances. These drops happen because deployed systems always face changes such as system upgrades and user behavior changes, which lead to mismatch between training and test data. It is particularly problematic for deployed applications where errors can result in system inefficiencies (e.g., VMCPU) or wasted operator time (e.g., NETIR).

(2) Drops in accuracy can lead to drastic increase in operational costs. The bottom of Fig. 1 shows the operational cost on a per-sample basis for these two systems. The cost of ML deployments come from three sources. First, some mistakes directly waste more resources (e.g., a mis-routed incident wastes time for innocent teams). Second, the system becomes less efficient when its ML model is not very accurate (e.g., the VM resource allocator needs to use a more conservative policy to account for inaccurate ML predictions). Third, the system may need to disable the ML solution entirely when its ML accuracy is not stable (blue bars in Fig. 1a), and its non-ML, human-based solution is inherently inefficient and expensive: even though the non-ML solution is more accurate than unstable ML models, human decision-making results in delay in routing the incidents and increased customer impact. Hence, drops in accuracy can lead to *superlinear* increase in cost.

A natural idea to solve this problem is to use more sophisticated data drift solutions in the ML literature (§2.1). However, these solutions have fundamental limitations for large-scale systems, such as scalability, delay in obtaining ground truths, and mixed data drifts (§1). §5 evaluates these ML solutions, and the results show that they are still inadequate for our deployed systems.

Based on our study of our deployed applications, a solution to the data drift problem in practice needs to satisfy the following properties:

- **Scalability:** The system needs to be able to handle millions of incoming test samples per second, run inference for those samples in real-time, and have reasonable memory and CPU overheads.
- **Adaptability:** The system needs to adapt to data drift even if the latest ground truth labels (for the most recent test samples) are not yet available.
- **Flexibility:** The system needs to be able to work with any type of predictive model, data types, and mixed types of data drifts.

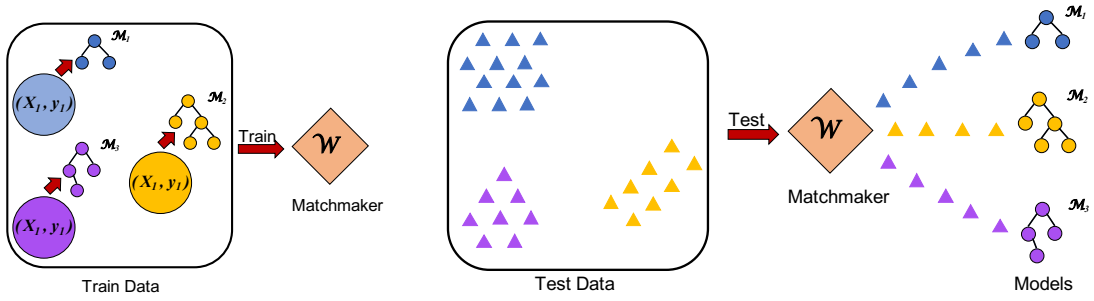
Summary of findings: We find that data drifts significantly impact deployed applications in practice and that the nature of the problem precludes many sophisticated approaches in the ML literature. We observe that any solution to the data drift problem has to satisfy three key properties: scalability, availablility, and flexibility. These form our design requirements when developing Matchmaker.

3 Overview of Matchmaker

We present an overview of our system - Matchmaker (Figure 2), an adaptive and flexible solution to the data drift problem that can run at scale. A summary of frequently used notation is provided in Table 1.

Symbol	Meaning
t	Batch index
T	Number of training batches
$(\mathbf{X}_t, \mathbf{y}_t)$	(Points,labels) in batch t
$(\mathbf{x}, y)$	One (point,label) pair
B_t	Number of points in batch t
$\mathcal{M}_t$	Model trained on batch t
$\mathcal{R}$	Random Forest for data partitioning
argsort $\mathcal{S}$	Sort elements of set $\mathcal{S}$ in *descending order*

Table 1. Frequently used notation and associated meanings.



(a) Training (Offline) (b) Testing (Online)

Figure 2. Overview of Matchmaker with 3 training batches. Predictive models ($\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$) are trained offline along with Matchmaker, $\mathcal{W}$ (Fig. 2a). At test time (Fig. 2b) $\mathcal{W}$ assigns each test point to the model from the most similar batch (same color) for prediction.

Matchmaker addresses the two main causes of data drift - covariate shift and concept drift (§2.1). The key idea behind Matchmaker is to *identify the batch of training data that is most similar to a new test sample and use the model trained on that batch for inference*. We rely on the following two conjectures for this:

P1 If the accuracy drop occurs because training and test data lie in different parts of the data space (*covariate shift*), then it makes sense to give more importance to the batch t of training data whose features ($\mathbf{X}_t$) are *nearest* to the features ($\mathbf{x}_*$) of the test data.

P2 If the accuracy drop occurs because of a change in $\mathbf{x} \rightarrow y$ relationship over time (*concept drift*), then it makes sense to give more importance to the batch of training data that would best reflect the current $\mathbf{x} \rightarrow y$ relationship (Brzezinski & Stefanowski, 2013).

Ideally in **P2**, we would like to give more importance to the batch t of training data whose $\mathbf{X}_t \rightarrow y$ mapping is *nearest* to that of the test data. However, it is not possible to compare mappings at test time since test labels are not available then. So, we follow prior work in assuming that the most recent batch best reflects the current $\mathbf{x} \rightarrow y$ relationship.

Matchmaker operates as follows:

1. **Train models:** We train a model for each batch of training data (i.e., T models for T batches).
2. **Compute covariate shift ranking:** We train a random forest, $\mathcal{R}$, on all labeled training batches $\{(\mathbf{X}_1, \mathbf{y}_1), \ldots, (\mathbf{X}_T, \mathbf{y}_T)\}$ which can partition the space of training samples (Davies & Ghahramani, 2014). At test time, we rank training batches based on *spatial nearness* to the test sample by identifying the leaf nodes that a test sample is mapped to in $\mathcal{R}$ and then rank the training batches in terms of the number of points from each batch in those leaf nodes (§4.1).
3. **Compute concept drift ranking:** Batches are ranked in decreasing order of accuracy of their models on the most recent batch (§4.2).
4. **Combine rankings:** The covariate shift and concept drift rankings for each test sample are combined using the Borda Count method (Shah & Wainwright, 2017).
5. **Predict label:** The model corresponding to the highest ranked batch is deployed on the test sample.

Performance properties. The computed concept drift ranking of the training batches $\{(\mathbf{X}_1, \mathbf{y}_1), \ldots, (\mathbf{X}_T, \mathbf{y}_T)\}$ is stored as a single vector while the covariate shift ranking of batches at each leaf node is computed and stored in a lookup table (see Algorithm 1). Hence, given traversal paths of a test sample, we can use a single lookup to the table to retrieve the covariate shift ranking at test time (see Algorithm 2) and combine this with the concept drift ranking to output the final ranking of training batches for the test sample. The ranking adds minimal overhead while inference itself is done using a *single* (highest ranked) model. Hence Matchmaker is significantly faster (*scalable*) compared to state-of-the art drift mitigation methods like AUE (Brzezinski & Stefanowski, 2013) which use an ensemble of models and have inference latency proportional to the size of the ensemble. The covariate shift ranking allows Matchmaker to mitigate data drifts when test samples come in without waiting for ground truth labels for model retraining (*adaptive*). Lastly Matchmaker does not depend on the class of models $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_T$ and can work with *any* model class, for eg. DNNs, SVMs, random forests, etc. (*flexible*).

4 Building Matchmaker

The main building block of Matchmaker is the algorithm for choosing from a set of pre-trained models during inference (Fig. 2). Each of these models $\mathcal{M}_1, \ldots, \mathcal{M}_T$ is trained using a different training set. In this work, we take these training sets to correspond to one batch in $(\mathbf{X}_1, \mathbf{y}_1), \ldots, (\mathbf{X}_T, \mathbf{y}_T)$. Thus we have T trained ML mod-

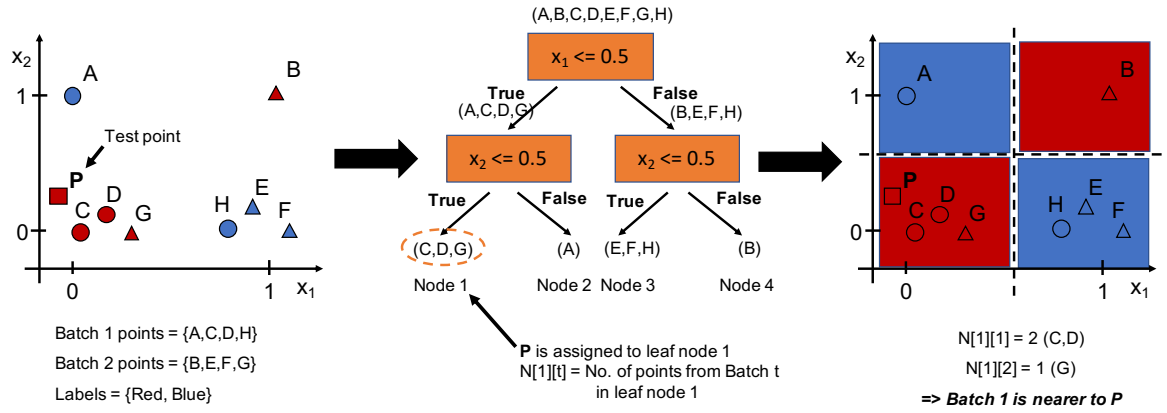


Figure 3. Partitioning of training sample space using Decision Trees. All points at a leaf node belong to the same class (red/blue).

els corresponding to T batches.

The goal of the model selection algorithm is to maximize prediction accuracy by using the model trained on the training set *most similar* to a test instance for inference. The algorithm consists of the following three components.

4.1 Covariate Shift Ranking

Our aim in ranking training sets in terms of covariate shift is based on **P1**. As per **P1** we should rank training batches in terms of nearness to the test point in the data space. A natural approach is to rank training batches in terms of average Euclidean distance from the test point. However, the Euclidean distance is expensive to compute (cost increases with batch size), sensitive to outliers (Fischler & Bolles, 1981) and susceptible to the curse of dimensionality (Xia et al., 2015) and thus may not be suitable for real-world settings with complex, high-dimensional data. Instead, in this section, we will present an alternate approach using decision trees, and its generalization to random forests, for ranking training batches in terms of nearness to test data. This approach is scalable, robust to outliers, and potentially less susceptible to the curse of dimensionality.

Ranking using decision trees. Decision trees organize data by splitting along features at thresholds chosen to maximize prediction accuracy (Breiman et al., 1984). Intuitively, decision trees *partition* the training data such that *similar* training samples are assigned to the same tree leaf and at test time the predicted label for a test point is typically the majority label of the leaf to which it is assigned. We leverage this property of decision trees in ranking training batches in terms of covariate shift (nearness to test point) as per **P1**.

Specifically, given a decision tree constructed using *all* training batches $(\mathbf{X}_1, \mathbf{y}_1), \ldots, (\mathbf{X}_T, \mathbf{y}_T)$, let $\mathbf{N}[k][t]$ denote the number of samples from batch t in leaf node k of the tree. For a test point assigned to leaf node k^*, the covariate shift ranking of training batches, r_{CS}, is in *decreasing order* of $\mathbf{N}[k^*][t]$ (lowest to highest covariate shift) as,

$$r_{\text{CS}} = \text{argsort}\,\{\mathbf{N}[k^*][1], \ldots, \mathbf{N}[k^*][T]\} \tag{1}$$

Fig. 3 illustrates this for a toy example. The key idea here is that, since data points at the same leaf node of a decision tree follow the same path through the tree and lie in nearby regions of the data space, a training batch with more samples in the leaf node that the test point is assigned to, can be considered to be *nearer* to it. (Davies & Ghahramani, 2014) use a similar idea to construct kernel functions.

We expect this approach to be *less sensitive to outliers* than the Euclidean distance since we are selecting batches based only on the points in a batch that are near (share a leaf node with) the test sample and are not considering points that are far away. An added benefit of decision trees is that they work well with a wide range of data types (categorical, integer, etc.) for which Euclidean distance may not be well-defined.

Generalized ranking using random forests. Random Forests (Breiman, 2001) are ensembles of decision trees that have demonstrated consistently higher prediction accuracy than individual trees (Fernández-Delgado et al., 2014) and work well in high-dimensional settings (Qi, 2012). We generalize our ranking approach to random forests to obtain improved performance especially on *high dimensional data* (Appendix C.2 has an ablation study to support this claim).

As discussed above, a decision tree can *rank training batches* in terms of the number of points in the batch that lie in the same leaf node. We train a random forest $\mathcal{R}$ on *all* training batches $(\mathbf{X}_1, \mathbf{y}_1), \ldots, (\mathbf{X}_T, \mathbf{y}_T)$. Each tree in $\mathcal{R}$ outputs a distinct ranking of training batches. We combine these rankings using the well known Borda Count method (Borda, 1784; Shah & Wainwright, 2017) to obtain a single ranking. For every ranking, the Borda count method assigns a score to each batch equal to the number of batches ranked below it. The combined ranking of the batches is

Algorithm 1 Matchmaker Training (Offline)

Input : Training batches: $(\mathbf{X}_1, \mathbf{y}_1), \ldots, (\mathbf{X}_T, \mathbf{y}_T)$
Hyperparameters for random forest $\mathcal{R}$
Output : Trained models: $\mathcal{M}_1, \ldots, \mathcal{M}_T$
Matchmaker random forest $\mathcal{R}$

Train $\mathcal{R}$ on entire data $\{(\mathbf{X}_1, \mathbf{y}_1), \ldots, (\mathbf{X}_T, \mathbf{y}_T)\}$
for *Tree* $\mathcal{T}_i \in \mathcal{R}$ **do**
  Store $\mathbf{S}^i[k^i][t] = \sum_{t' \neq t} \mathbb{1}\left\{\mathbf{N}[k^i][t] > \mathbf{N}[k^i][t']\right\}$
end
for *Batch* $t \in T$ **do**
  Train model $\mathcal{M}_t$ on data batch $(\mathbf{X}_t, \mathbf{y}_t)$
  $\mu_t = 1 - \frac{1}{B_t}\left(\sum_{(\mathbf{x},y)\in(\mathbf{X}_T,\mathbf{y}_T)}(1 - \mathcal{M}_t(\mathbf{x})[y])^2\right)$
end
Set $\mu_T = 1$
Concept Drift Ranking (r_{cd}) = argsort $\{\mu_1, \ldots, \mu_T\}$
Return $\mathcal{M}_1, \ldots, \mathcal{M}_T, \mathcal{R}$

Algorithm 2 Matchmaker Inference (Online)

Input : Trained models: $\mathcal{M}_1, \ldots, \mathcal{M}_T$
Matchmaker random forest $\mathcal{R}$
Test data point: $\mathbf{x}_*$
Output : Predicted label: $\hat{y}$

for *Tree* $\mathcal{T}_i \in \mathcal{R}$ **do**
  $k^i_* =$ Leaf node in $\mathcal{T}_i$ that $\mathbf{x}_*$ is mapped to
  $s^*_{it} = \mathbf{S}^i[k^i_*][t]$
end
for *Batch* $t \in T$ **do**
  $s_t = \sum_{i=1}^{n} s^*_{it}$
end
Covariate Shift Ranking (r_{cs}) = argsort $\{s_1, \ldots, s_T\}$
Final Ranking $(r_*) =$ Borda count $(r_{\text{cs}}, r_{\text{cd}})$
Set $\hat{t} = r_*[0]$
Return $\hat{y} = \mathcal{M}_{\hat{t}}(\mathbf{x}_*)$

in decreasing order of the total score for each batch (see Appendix A for a detailed description). This gives the final *covariate shift ranking*, r_{cs}, of training batches. Combining rankings in this fashion is potentially less sensitive to outliers than alternatives like averaging scores from each decision tree (Demšar, 2006). It also enables efficient computation of covariate shift ranking for test samples as Borda Count scores at each leaf node of every decision tree in $\mathcal{R}$ is stored in a lookup table (Algorithm 1) and the ranking is computed in decreasing order of the sum of scores from leaf nodes that a test point is assigned to (Algorithm 2).

4.2 Concept Drift Ranking

We now need to rank batches in terms of lowest to highest concept drift based on **P2**. Following prior work (Brzezinski & Stefanowski, 2013; Tahmasbi et al., 2020), we assume that the most recently received training batch $(\mathbf{X}_T, \mathbf{y}_T)$ has the least concept drift (most similar $\mathbf{x} \rightarrow y$ relationship) with respect to the test data. The remaining batches are ranked based on how close *their* $\mathbf{x} \rightarrow y$ relationship is to that of the most recent training batch. For this we compute a score for batches $1, \ldots, T-1$, analogous to the score in (Brzezinski & Stefanowski, 2013), given by

$$\mu_t = 1 - \frac{1}{B_t}\left(\sum_{(\mathbf{x},y)\in(\mathbf{X}_T,\mathbf{y}_T)} (1 - \mathcal{M}_t(\mathbf{x})[y])^2\right) \quad (2)$$

where B_t is the batch size and $\mathcal{M}_t(\mathbf{x})[y]$ denotes the probability that model $\mathcal{M}_t$ predicts the correct class (y) on any point $\mathbf{x}$. Thus if $\mathcal{M}_t$ assigns a low probability to the correct class of points in the most recent batch then the score μ_t will be low. We set $\mu_T = 1$ to ensure that the most recent batch is ranked first. The concept drift ranking of training batches, r_{cd} (lowest to highest concept drift), is given by,

$$r_{\text{cd}} = \text{argsort}\,\{\mu_1, \ldots, \mu_T\} \quad (3)$$

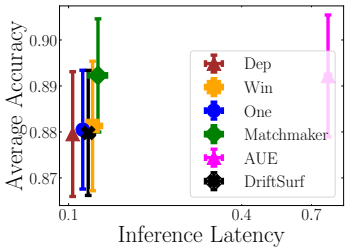
4.3 Final Ranking

Since Matchmaker outputs two rankings based on covariate shift and concept drift respectively, we combine them by again using the Borda Count method (see Appendix A) to obtain a *single* ranking of the batches that considers both kinds of data drift. This ranking is computed efficiently for each test point as described in Algorithm 2.

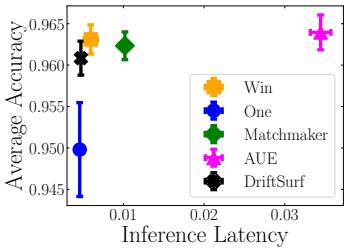
5 Evaluation

We evaluate Matchmaker on our production datasets - NETIR and VMCPU, two synthetic datasets - Covcon (a synthetic dataset of our construction with both covariate shift and concept drift) and Circle (Pesaranghader et al., 2016) (a synthetic dataset from prior work that only has concept drift) - as well as two publicly available datasets - Electricity (Zliobaite, 2013), and Covertype (Blackard & Dean, 1999) - with unknown drift that are commonly used to evaluate drift mitigation approaches. We use Python 3.6 for all experiments and use the random forest classifier from Python's Scikit Learn library (SKL) for the predictive models $(\mathcal{M}_1, \ldots, \mathcal{M}_T)$. We train and run Matchmaker as per Algorithms 1 and 2 with the number of retained batches (T) set to 7. The random forest, $\mathcal{R}$ used in Matchmaker, is constructed with squared error as the splitting criterion. We compare Matchmaker to the following baselines:

1. Win: Uses a predictive model trained on the past 7 batches for prediction on incoming data.
2. One: Uses a predictive model trained on only the *most recent* batch for prediction on incoming data.



(a) NetIR



(b) VMCPU

Method	NetIR	VMCPU
AUE	1.63 ± 0.24	**77.02 ± 3.64**
Dep	1.91 ± 0.26	N/A
DriftSurf	1.66 ± 0.24	86.65 ± 3.23
Matchmaker	**1.57 ± 0.23**	79.57 ± 3.24
One	1.69 ± 0.24	88.22 ± 4.49
Win	1.66 ± 0.24	80.45 ± 3.30

(c) Cost

Figure 4. Plots (a) and (b) are for Average Accuracy v/s Inference Latency for internal datasets (error bars correspond to one standard error). Higher (high accuracy) and to the left (low latency) is better for both plots. Table (c) contains average cost values for each approach on the internal datasets (lower is better).

3. AUE(Brzezinski & Stefanowski, 2013): Uses an ensemble of 7 models which are retrained after each batch of data is received. The voting weights for each model is determined by its accuracy on most recent batch.
4. DriftSurf(Tahmasbi et al., 2020): Uses an adaptive approach based on drift detection. The algorithm maintains a model trained on all the past data. Whenever a drift is detected a new model is trained on the data received after drift detection. The algorithm greedily switches between old and new models as long as data drift is being detected and retains the best performing model after the drift has passed.

For NETIR, we also compare with the baseline Dep, the currently deployed approach in production. It uses a model trained on *all* past batches where the batches are assigned linearly decaying weights (as per the weighted training procedure of (SKL)) for training, and where misclassified examples are assigned higher weight ($1.8\times$ higher). For VMCPU, Win is the approach currently deployed in production.

We use the following metrics for evaluation:

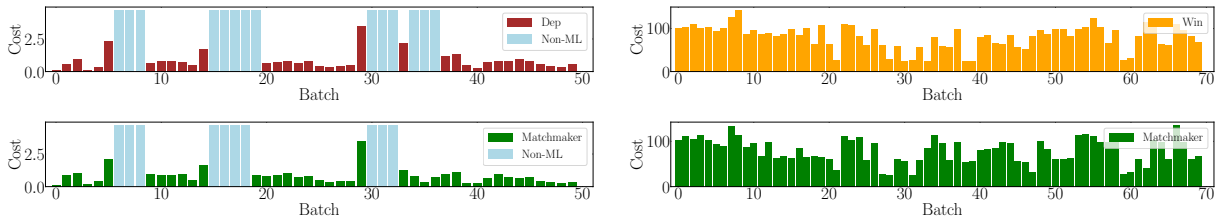
1. **Average Accuracy:** Average fraction of correct predictions for the approach (averaged over all batches)
2. **Inference Latency:** Average test time for the approach (averaged over first 100 points for 5 batches for VMCPU, and 10 batches for other datasets).
3. **Average Cost:** (Only for NETIR and VMCPU) Weighted sum of cost-per-sample of incorrect prediction (increases with errors), and cost-per-sample of running inference (increases with inference overhead) using the approach. It also includes the cost of running a default (non-ML) solution which is deployed if the accuracy in any batch drops below 75% of the maximum possible accuracy and is deployed until the accuracy of the ML solution rises to an acceptable level (see Appendix B for details).

5.1 Performance on Internal Datasets

The performance of Matchmaker on the internal datasets is summarised in Fig. 4 while more fine-grained plots in Fig. 5 show the cost-per-batch for Matchmaker and the deployed model for each dataset (Dep for NETIR and Win for VMCPU).

Fig. 4a shows that Matchmaker has the *highest* average accuracy for the NETIR dataset and is *significantly* (almost $8\times$) faster than AUE, the only other approach with comparable accuracy. Additionally the table in Fig. 4c shows that Matchmaker has the *least cost* for NETIR and clearly outperforms AUE and DriftSurf, the two state-of-the-art methods for drift mitigation. Finally, the plots in Fig. 5a shows that Matchmaker has a significantly lower cost than Dep on several batches since it avoids deploying the expensive non-ML solution (involving prediction by human experts) on many batches. Due to its clear overall superiority, we are currently in the process of deploying Matchmaker in production for NETIR.

For VMCPU, we see from the table in Fig. 4c that Matchmaker has a lower cost than the currently deployed model Win and the state-of-the-art drift mitigation method DriftSurf. Only AUE has a lower cost than Matchmaker on this dataset but Matchmaker is almost $4\times$ faster than AUE on this dataset (Fig. 4b). Interestingly Win has a slightly higher average accuracy than Matchmaker on this dataset but Matchmaker has a lower cost since the cost function (see Appendix B) assigns different weights to false positives and false negatives and Matchmaker performs better on the balance. However due to the somewhat conflicting results we conclude that there is no clear winner between Matchmaker, AUE, and Win for VMCPU and so are continuing to deploy Win while exploring further options for improving the performance of Matchmaker on this dataset.



(a) NETIR (b) VMCPU

Figure 5. Normalized operational cost per batch on internal datasets - NETIR and VMCPU. Observe how matchmaker *clearly* saves on cost in NETIR by avoiding the expensive Non-ML solution in many batches.

5.2 Performance on External Datasets

We evaluate Matchmaker and the baseline methods on two synthetic datasets, and two publicly available (real-world) dataset that are commonly used for evaluating drift mitigation approaches (Tahmasbi et al., 2020). Appendix C.1 describes the details of these datasets.

We construct the synthetic dataset Covcon to have both covariate shift (batches are drawn from different parts of the data space) and concept drift (the $\mathbf{x} \rightarrow y$ relationship changes over time). The other synthetic dataset Circle (Pesaranghader et al., 2016) has only concept drift. Not surprisingly, Matchmaker outperforms *all* baselines for Covcon (Fig. 7c) since it can handle both covariate shift and concept drift while the baselines are more suited for handling only concept drift. This also offers a potential explanation for the slightly better performance of AUE and DriftSurf for Circle (Fig. 7d), which has only concept drift. However note that even in this case Matchmaker has only slightly lower accuracy than these two baselines and continues to have much lower latency than AUE. These results strengthen our claim that Matchmaker is preferable in settings where there are both covariate shift and concept drift.

For public datasets Electricity (Fig. 7e) and CoverType (Fig. 7f), most methods show comparable performance. Indeed, (Pesaranghader et al., 2018) notes that there seems to be consensus among researchers (Frias-Blanco et al., 2014; Pesaranghader & Viktor, 2016; Huang et al., 2015; Bifet et al., 2009; Losing et al., 2018) that it is unclear if there is any actual data drift in these datasets and thus there may not be any need for deploying drift mitigation approaches. However, our results do show that deploying Matchmaker will not incur much additional cost while maintaining competitive accuracy. Interestingly we observe that One, which only uses the most recent batch of data, outperforms all baselines for CoverType. While the reason is not entirely clear, we believe that One's high variability across datasets and its poor performance on internal datasets NETIR and VMCPU makes it unsuitable for real-world deployment.

Additional results. We also include an ablation study in Appendix C.2 where we show that 1) The design choices (covariate shift ranking using random forests and concept drift ranking based on accuracy) lead to tangible accuracy gains, and 2) Matchmaker's perforamnce does not vary significantly across different hyperaparameter settings. Appendix C.3 shows results with additional baselines.

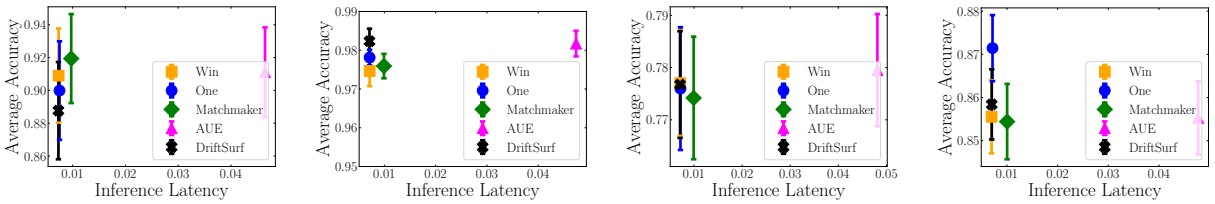
6 Discussion

We discuses several alternative designs, limitations, and future directions for Matchmaker.

(1) Intelligent training batch construction. We currently construct training data batch according to the arrival time of ground truth labels. An alternative is to construct batches by cluster points according to some similarity metric which may better capture local concepts. The potential downside is that there will be reduced diversity in each training batch, and the corresponding ML model may have an overfitting problem. We leave this exploration as a future direction.

(2) Fine-tune base models using AutoML. One of the potential approaches to boost accuracy is to leverage AutoML systems (e.g., (Feurer et al., 2019; Google Cloud AutoML; Azure Automated Machine Learning; SageMaker Autopilot)) to select the best ML model and hyperparameters for each training data batch. Our evaluation using this approach alone (i.e., without Matchmaker) yields only small improvements on accuracy ($\leq 2\%$) while increasing the training overheads by two orders of magnitude. This approach might work better if we construct training data batch based on similarity, and we plan to explore this direction further.

(3) Unstructured data. Our current design uses a random forest to determine data similarity. This works well for structured data (e.g., tables), which are widely used in ML optimizations for systems. However, it may not work well for high-dimensional, unstructured data such as images or text. A potential approach for such data is to leverage out-of-distribution detection methods using deep neural networks



(a) Covcon (b) Circle (c) Electricity (d) Covertype

Figure 6. Average Accuracy v/s Inference Latency for external datasets. Error bars correspond to one standard error.

(DNNs) (Hendrycks & Gimpel, 2017; Liang et al., 2018).

7 Related Work

To the best of our knowledge, Matchmaker is the first scalable data drift solution that is capable of handling both covaraite shift and concept drift in the same test batch without waiting for next round of model retraining. We expand our discussion on related work here.

Covariate shift. In the ML literature, covariate shift (Shimodaira, 2000), virtual drift (Tsymbal, 2004), and population shift (Kelly et al., 1999) all refer to the same type of data drift, where the training data distribution does not match the test data distribution while sharing similar underlying mapping from features to labels. The vast majority of work in this field only concerns two datasets (training and test) and there is no notion of continuing time (Ditzler et al., 2015). Hence, popular solutions treat this as a learning problem by adjusting training objectives or training data importance to achieve better test data accuracy (e.g., (Shimodaira, 2000; Sugiyama et al., 2007; Yamazaki et al., 2007; Mansour et al., 2008; Gretton et al., 2009; Pan & Yang, 2010)). The closet to our work is a recent drift adaptation solution for video analytics, ODIN (Suprem et al., 2020), which picks the best-fit specialized models to process input video frames using generative adversarial networks (GANs) (Goodfellow et al., 2014). In contrast, our solution is a better fit to ML for large-scale systems because it (1) is not specific to video analytics; (2) does not involve expensive DNNs; and (3) can handle both covariate shift and concept drift.

Concept drift. When the test data changes its feature-to-label mapping from the training data, the data drift problem becomes concept drift or real drift (Tsymbal, 2004). This problem is fundamentally different from covariate shift because training data under obsolete concepts is harmful to model performance, and a concept drift solution needs to determine which training data to *forget*. Concept drift solutions mostly fall into three categories (Ditzler et al., 2015): (1) *window-based solutions* that train models using a sliding window or assigning higher importance to recent data points (e.g., (Widmer & Kubat, 1996; Koychev, 2000; Hentschel et al., 2019; Klinkenberg, 2004)), which are also the ones used in our deployed systems; (2) *detection methods* that detects concept drifts and generates new ML models accordingly (e.g., (Kifer et al., 2004; Gama et al., 2004; Bifet & Gavaldà, 2007; Pesaranghader et al., 2018; Tahmasbi et al., 2020)); and (3) *ensemble methods* that maintains a collection of ML models and continuously update the ML models and their voting weights (e.g. (Elwell & Polikar, 2011; Brzezinski & Stefanowski, 2013; Sun et al., 2018; Zhao et al., 2020)). The major drawback of these concept drift solutions is that their adaptation only happens after new batch of ground truth is available, which is not agile enough for ML deployments at large scale. As we show in our evaluation (§5), Matchmaker addresses the data drift problems much better than window-based and detection methods. While ensemble methods such as AUE are more competitive in model accuracy, their high overheads make them unappealing for large-scale systems.

8 Conclusion

As ML techniques are widely deployed in various aspects of modern data centers, accuracy drops caused by data drift can lead to significant system inefficiencies. Our study on two real-world ML deployments demonstrates that ML models are vulnerable to data drifts despite frequent model retraining. We present Matchmaker, the first scalable, adapatable, and flexible solution to the data drift problem for ML deployments in large-scale systems. Matchmaker addresses real-world problems that existing ML solutions do not address, such as scalability, ground truth latency, and mixed types of data drifts. Matchmaker operates by finding the most *similar* training data batch for each test point and using the corresponding ML model for inference. As part of our solution, we introduce a novel similarity metric to address multiple types of data drifts with low computation overheads. Our evaluation using real-world ML deployments demonstrate that Matchmaker yields significant reductions in accuracy variation, while providing much faster ML inference than state-of-the-art ML data drift solutions.

REFERENCES

Scikit-learn random forest. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

Abu-Libdeh, H., Altinbüken, D., Beutel, A., Chi, E. H., Doshi, L., Kraska, T., Li, X., Ly, A., and Olston, C. Learned indexes for a google-scale disk-based database. *CoRR*, abs/2012.12501, 2020.

Arzani, B., Ciraci, S., Saroiu, S., Wolman, A., Stokes, J. W., Outhred, G., and Diwu, L. PrivateEye: Scalable and privacy-preserving compromise detection in the cloud. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

Azure Automated Machine Learning. https://azure.microsoft.com/en-us/services/machine-learning/automatedml/.

Balaji, B., Kakovitch, C., and Narayanaswamy, B. FirePlace: Placing firecracker virtual machines with hindsight imitation. In *NeurIPS Workshop on Machine Learning for Systems*, 2020.

Baylor, D., Haas, K., Katsiapis, K., Leong, S., Liu, R., Menwald, C., Miao, H., Polyzotis, N., Trott, M., and Zinkevich, M. Continuous training for production {ML} in the tensorflow extended ({TFX}) platform. In *2019 {USENIX} Conference on Operational Machine Learning (OpML 19)*, pp. 51–53, 2019.

Bifet, A. and Gavaldà, R. Learning from time-changing data with adaptive windowing. In *Proceedings of the SIAM International Conference on Data Mining (ICDM)*, 2007.

Bifet, A. and Gavalda, R. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*, pp. 443–448. SIAM, 2007.

Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., and Gavalda, R. New ensemble methods for evolving data streams. In *Proceedings of the international conference on Knowledge discovery and data mining (KDD)*, pp. 139–148, 2009.

Blackard, J. A. and Dean, D. J. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 24(3):131–151, 1999.

Borda, J. d. Mémoire sur les élections au scrutin. *Histoire de l'Academie Royale des Sciences pour 1781 (Paris, 1784)*, 1784.

Breiman, L. Random forests. *Machine learning*, 45(1), 2001.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. *Classification and regression trees*. CRC press, 1984.

Brzezinski, D. and Stefanowski, J. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1), 2013.

Chowdhury, G. G. Natural language processing. *Annual review of information science and technology*, 37(1), 2003.

Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., and Bianchini, R. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Symposium on Operating Systems Principles (SOSP)*, 2017.

Davies, A. and Ghahramani, Z. The random forest kernel and other kernels for big data from random partitions. *arXiv preprint arXiv:1402.4293*, 2014.

Demšar, J. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

Ditzler, G., Roveri, M., Alippi, C., and Polikar, R. Learning in nonstationary environments: A survey. *IEEE Comput. Intell. Mag.*, 10(4), 2015.

Elwell, R. and Polikar, R. Incremental learning of concept drift in nonstationary environments. *IEEE Trans. Neural Networks*, 22(10), 2011.

Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. Do we need hundreds of classifiers to solve real world classification problems? *The journal of machine learning research (JMLR)*, 15(1), 2014.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., and Hutter, F. Auto-sklearn: efficient and robust automated machine learning. In *Automated Machine Learning*, pp. 113–134. 2019.

Fischler, M. A. and Bolles, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6), 1981.

Flynn, J., Broxton, M., Debevec, P., DuVall, M., Fyffe, G., Overbeck, R., Snavely, N., and Tucker, R. Deepview: View synthesis with learned gradient descent. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2367–2376, 2019.

Frias-Blanco, I., del Campo-Ávila, J., Ramos-Jimenez, G., Morales-Bueno, R., Ortiz-Diaz, A., and Caballero-Mota, Y. Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):810–823, 2014.

Gama, J., Medas, P., Castillo, G., and Rodrigues, P. P. Learning with drift detection. In *Brazilian Symposium on Artificial Intelligence (SBIA)*, 2004.

Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4), 2014.

Gao, J. Machine learning applications for data center optimization. 2014.

Gao, J., Yaseen, N., MacDavid, R., Frujeri, F. V., Liu, V., Bianchini, R., Aditya, R., Wang, X., Lee, H., Maltz, D., et al. Scouts: Improving the diagnosis process through domain-customized incident routing. In *Proceedings of the Annual conference of the Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM)*, 2020.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. Generative adversarial nets. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2014.

Google. Google cloud incidents. https://status.cloud.google.com/summary.

Google Cloud AutoML. https://cloud.google.com/automl.

Gretton, A., Smola, A., Huang, J., Schmittfull, M., Borgwardt, K., and Schölkopf, B. Covariate shift by kernel mean matching. *Dataset shift in machine learning*, 3(4), 2009.

Hendrycks, D. and Gimpel, K. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

Hentschel, B., Haas, P. J., and Tian, Y. Online model management via temporally biased sampling. *SIGMOD Record*, 48(1), 2019.

Hossain, M. S., Rahaman, S., Kor, A., Andersson, K., and Pattinson, C. A belief rule based expert system for datacenter PUE prediction under uncertainty. *IEEE Trans. Sustain. Comput.*, 2(2), 2017.

Huang, D. T. J., Koh, Y. S., Dobbie, G., and Bifet, A. Drift detection using stream volatility. In *Joint European conference on machine learning and knowledge discovery in databases*, 2015.

Kelly, M. G., Hand, D. J., and Adams, N. M. The impact of changing populations on classifier performance. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 1999.

Kifer, D., Ben-David, S., and Gehrke, J. Detecting change in data streams. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2004.

Klinkenberg, R. Learning drifting concepts: Example selection vs. example weighting. *Intell. Data Anal.*, 8(3), 2004.

Kolter, J. Z. and Maloof, M. A. Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research (JMLR)*, 8, 2007.

Koychev, I. Gradual forgetting for adaptation to concept drift. Proceedings of ECAI Workshop on Current Issues in Spatio-Temporal Reasoning, 2000.

Lazic, N., Boutilier, C., Lu, T., Wong, E., Roy, B., Ryu, M. K., and Imwalle, G. Data center cooling using model-predictive control. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4), 1989.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research (JMLR)*, 18(1), 2017.

Liang, S., Li, Y., and Srikant, R. Enhancing the reliability of out-of-distribution image detection in neural networks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

Losing, V., Hammer, B., and Wersing, H. Incremental online learning: A review and comparison of state of the art algorithms. *Neurocomputing*, 275, 2018.

Mansour, Y., Mohri, M., and Rostamizadeh, A. Domain adaptation with multiple sources. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L. (eds.), *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2008.

McDiarmid, C. On the method of bounded differences. *Surveys in combinatorics*, 141(1), 1989.

Moreno-Torres, J. G., Raeder, T., Alaiz-Rodríguez, R., Chawla, N. V., and Herrera, F. A unifying view on dataset shift in classification. *Pattern recognition*, 45(1):521–530, 2012.

Pan, S. J. and Yang, Q. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.*, 22(10), 2010.

Pesaranghader, A. and Viktor, H. L. Fast hoeffding drift detection method for evolving data streams. In *Joint European conference on machine learning and knowledge discovery in databases*, 2016.

Pesaranghader, A., Viktor, H. L., and Paquet, E. A framework for classification in data streams using multi-strategy learning. In *International conference on discovery science*, pp. 341–355. Springer, 2016.

Pesaranghader, A., Viktor, H. L., and Paquet, E. Mcdiarmid drift detection methods for evolving data streams. In *International Joint Conference on Neural Networks (IJCNN)*, 2018.

Qi, Y. Random forest for bioinformatics. In *Ensemble machine learning*, pp. 307–323. Springer, 2012.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., and Li, F. ImageNet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 115(3), 2015.

Rzadca, K., Findeisen, P., Swiderski, J., Zych, P., Broniek, P., Kusmierek, J., Nowak, P., Strack, B., Witusowski, P., Hand, S., and Wilkes, J. Autopilot: workload autoscaling at Google. In *Proceedings of ACM European Conference on Computer Systems (EuroSys)*, 2020.

SageMaker Autopilot. https://aws.amazon.com/sagemaker/autopilot/.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., and Young, M. Machine learning: The high interest credit card of technical debt. In *SE4ML: Software Engineering for Machine Learning (NeurIPS Workshop)*, 2014.

Shah, N. B. and Wainwright, M. J. Simple, robust and optimal ranking from pairwise comparisons. *The Journal of Machine Learning Research*, 18(1), 2017.

Shimodaira, H. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2), 2000.

Song, Z., Berger, D. S., Li, K., and Lloyd, W. Learning relaxed belady for content distribution network caching. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

Sugiyama, M., Krauledat, M., and Müller, K. Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research (JMLR)*, 8, 2007.

Sun, Y., Tang, K., Zhu, Z., and Yao, X. Concept drift adaptation by exploiting historical knowledge. *IEEE Trans. Neural Networks Learn. Syst.*, 29(10), 2018.

Suprem, A., Arulraj, J., Pu, C., and Ferreira, J. Odin: Automated drift detection and recovery in video analytics. *Proceedings of the VLDB Endowment*, 13(11).

Suprem, A., Arulraj, J., Pu, C., and Ferreira, J. E. ODIN: automated drift detection and recovery in video analytics. *Proceedings of the VLDB Endowment*, 13(11), 2020.

Tahmasbi, A., Jothimurugesan, E., Tirthapura, S., and Gibbons, P. B. Driftsurf: A risk-competitive learning algorithm under concept drift. *arXiv preprint arXiv:2003.06508*, 2020.

Tsymbal, A. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106(2), 2004.

Widmer, G. and Kubat, M. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23 (1), 1996.

Xia, S., Xiong, Z., Luo, Y., Zhang, G., et al. Effectiveness of the euclidean distance in high dimensional spaces. *Optik*, 126(24):5614–5619, 2015.

Yamazaki, K., Kawanabe, M., Watanabe, S., Sugiyama, M., and Müller, K. Asymptotic bayesian generalization error when training and test distributions are different. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2007.

Zhang, S., Meng, W., Bu, J., Yang, S., Liu, Y., Pei, D., Xu, J., Chen, Y., Dong, H., Qu, X., and Song, L. Syslog processing for switch failure diagnosis and prediction in datacenter networks. In *Proceedings of the IEEE/ACM International Symposium on Quality of Service (IWQoS)*, 2017.

Zhao, P., Cai, L., and Zhou, Z. Handling concept drift via model reuse. *Machine learning*, 109(3), 2020.

Zliobaite, I. How good is the electricity benchmark for evaluating concept drift adaptation. *arXiv preprint arXiv:1301.3524*, 2013.

A Overview of Borda Count

We use the Borda count algorithm (Borda, 1784; Shah & Wainwright, 2017) at two places in Matchmaker - first to combine the training batch rankings from the different trees in the random forest $\mathcal{R}$ to obtain the final covariate shift ranking, r_{cs}, and second to combine the covariate shift ranking, r_{cs}, and the concept drift ranking, r_{cd}, to obtain the final ranking. Here we explain the Borda count algorithm for combining a given set of rankings.

The Borda count ranking algorithm is a ranked voting system that is designed to combine rankings from multiple sources. Given a set of T items (training data batches, in our case) and n different rankings (or orderings) of the items, it operates as follows:

1. For item t in ranking i, calculate Borda Score s_{it}: the number of items that t beats in ranking i
2. Compute $s_t = \sum_{i=1}^{n} s_{it}^*$, total Borda Score for item t
3. Obtain the final ranking by sorting s_t from highest to lowest.

The following example illustrates the Borda count approach. Consider items {A,B,C,D} and two associated rankings, $\mathbf{r_1}$: {A,B,C,D}, and $\mathbf{r_2}$: {C,A,B,D}. The item-wise scores would be $s_A = 3 + 2 = 5$, $s_B = 2 + 1 = 3$, $s_C = 1 + 3 = 4$, and $s_D = 0 + 0 = 0$. Thus the final ranking is $\mathbf{r}_*$:{A,C,B,D} and the highest ranked item is A.

The Borda count approach makes no assumptions about how the underlying scores/rankings are obtained and easily scales: both properties that Matchmaker requires. (Shah & Wainwright, 2017) provides theoretical guarantees for this approach.

B Cost Function

The operational cost consists of four components:

1. **Mis-prediction cost.** The additional costs that are incurred by mis-predictions from ML models. For NetIR, this is the time for an incident spent in the teams that are not responsible for the incident. For VMCPU, this is the cost of additional host VMs for all the false positive predictions (a low-CPU VM that is incorrectly classified as a high-CPU VM, which lost its opportunity to save hardware resources).
2. **System adjustment cost.** The additional costs for the system to adjust its policy to meet system objectives. This only applies to VMCPU, where the VM resource allocator needs to adjust its policy based on the expected number of false negatives and false positives so that the probability of co-locating high-CPU VMs is below an acceptable threshold.
3. **Non-ML solution cost.** The costs of using the default, non-ML solution when the ML model accuracy drops below 75% of the maximum possible accuracy. Once the non-ML solution is deployed, the system will only enable the ML solution after three consecutive batches with acceptable accuracy.
4. **Inference cost.** The costs of running inference for the data drift solution based on the CPU time and the number of test instances in a data batch.

C Additional Evaluation Details

C.1 Details of External Datasets

We reported results on the following external datasets (in addition to our internal datasets - NETIR and VMCPU) in Section 5 (Figure 6). We simulate the arrival of batches over time by splitting each dataset into a sequence of equal-sized batches.

1. **Covcon**: We construct this 2-dimensional dataset to have covariate shift and concept drift. The decision boundary at each point is given by $\alpha * \sin(\pi x_1) > x_2$. We use 10000 points (100 batches, 1000 points per batch). Covariate shift is introduced by changing the location of x_1 and x_2 (for batch t x_1 and x_2 are drawn from the Gaussian distribution with mean $((t+1)\%7)/10$ and standard deviation 0.015). Concept drift is introduced by alternating the value of α between 0.8 and 1, every 25 batches and also changing the inequality from $>$ to $<$ after 50 batches.

2. **Circle** (Pesaranghader et al., 2016): This dataset contains two features x_1, x_2 drawn uniformly from the interval $[0, 1]$. We use 100000 points (100 batches, 1000 points per batch) from this dataset. Each data point is labeled as per the condition $(x_1 - c_1)^2 + (x_2 - c_2)^2 <= r$ where the center $(c_1, c_2))$ and radius r of the circular decision boundary changes gradually over a period of time introducing (gradual) concept drift. Specifically we change the center and radius at batch 25, 50, and 75. Each time the change happens gradually over the next 5 batches with the center and radius changing for 20% of the points each time.

3. **Electricity** (Zliobaite, 2013): This is a publicly available 13− dimensional dataset of the records of the New South Wales Electricity Market in Australia. The class labels represent change in price (Up/Down). There may be data drift due to change in consumption behaviour. We use 45312 points (34 batches, 1333 points per batch) from this dataset.

4. **Covertype** (Blackard & Dean, 1999): This is a multi-class dataset with 54 attributes and 581000 points (100 batches, 5810 points per batch) describing 7 forest cover types (7 classes) for 4 regions in the Roosevelt National Forest of Northern Colorado, obtained from US Forest Service (USFS) information system.

C.2 Ablation Study

Approach	Accuracy
Tree	0.872 ± 0.012
Forest	0.890 ± 0.011
Matchmaker	0.892 ± 0.012

Table 2. Accuracy of Matchmaker and its reduced versions - Tree and Forest on NETIR

Window Length	Number of Trees	Tree Depth
0.89 ± 0.01 (3)	0.90 ± 0.01 (25)	0.89 ± 0.01 (10)
0.90 ± 0.01 (**7**)	0.90 ± 0.011 (**50**)	0.90 ± 0.01 (**20**)
0.90 ± 0.01 (11)	0.90 ± 0.01 (75)	0.89 ± 0.01 (30)
0.89 ± 0.01 (15)	0.90 ± 0.01 (100)	0.90 ± 0.01 (40)

Table 3. Accuracy of Matchmaker on NETIR for different hyperparemeters. Bold values are the ones used in evaluation in Section 5

We run two ablation studies on the NETIR dataset to validate the design choices in Matchmaker:

1. To validate the ranking approach described in Section 4, we show in Table 2 that ranking batches using a single decision tree (Tree), or using a random forest but without the concept drift ranking (Forest), gives lower accuracy than Matchmaker which combines the covariate shift ranking obtained from the random forest with the concept drift ranking from the prediction performance of past models. This shows that the generalization of the ranking to random forests, and the incorporation of the concept drift ranking are *both* needed to obtain the desired accuracy gains.

2. Since ML models are often sensitive to the choice of hyperparameters (Li et al., 2017), we checked if the same holds for Matchmaker by sweeping over the 3 main hyperparameters of our model: (a) Number (T) of training batches retained, (b) number of trees in the Random Forest $\mathcal{R}$, and (c) maximum depth of the Random Forest $\mathcal{R}$. We present the results in Table 3 for the NETIR application. Our random forest for data partitioning consists of 50 trees and has a max depth of 20. We mark these choices with an asterisk ($*$). We find that the accuracy is not significantly affected by the choice of hyperparameters and all choices demonstrate superior accuracy than the baselines in Fig. 5a .

C.3 Additional Results

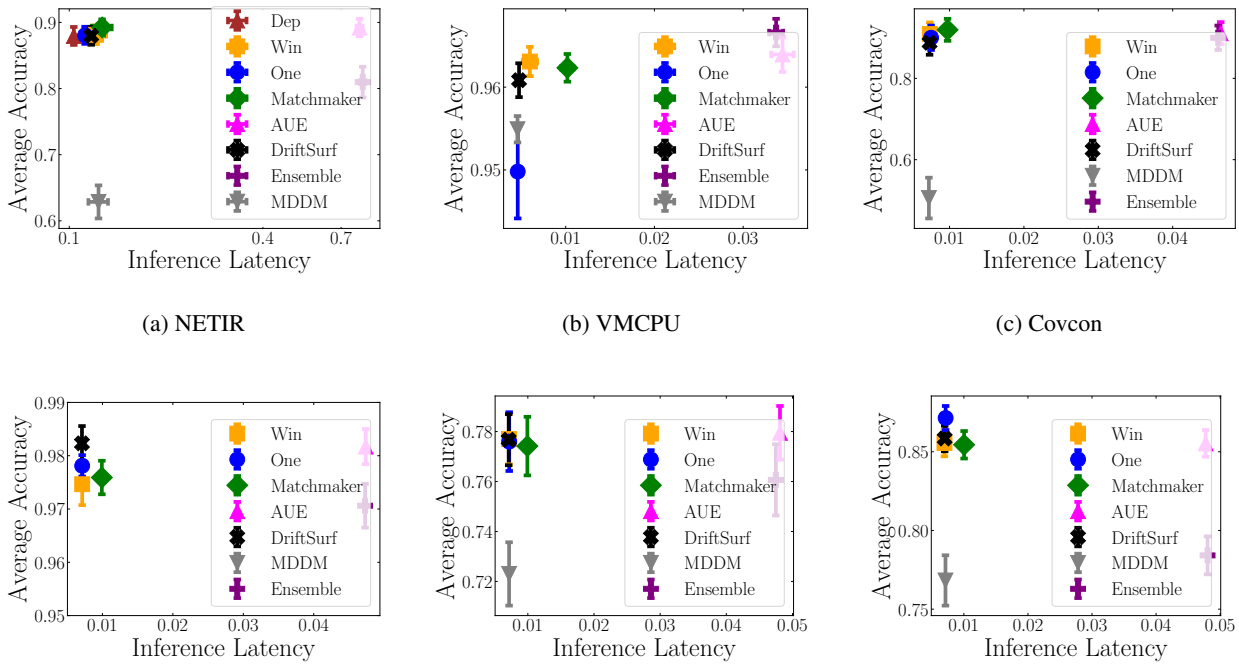
We present additional results in this section using two additional baselines:

1. MDDM(Pesaranghader et al., 2018): uses McDarmid's Inequality (McDiarmid, 1989) on batch accuracies to detect data drift and retrains the predictive model on the current batch if drift is detected.

2. Ensemble: uses an ensemble of 7 models trained the past 7 batches (one model per batch). The output is the average of individual model predictions

The results of accuracy v/s inference latency with these two new baselines for all datasets are presented in Fig. 7. MDDM generally performs poorly because we apply it on batch accuracies, unlike the per-sample accuracies used in (Pesaranghader et al., 2018) since true sample labels cannot be obtained in real-time in our setting. Ensemble generally has lower accuracy than AUE, which also uses an ensemble of models for prediction, and has similar inference latency as AUE, which is *much larger* than Matchmaker or the other baselines thus rendering it unsuitable for our latency critical applications.

For the sake of completeness, we also include plots of accuracy and cost for each batch for NETIR (Figures 8 and 9) and VMCPU (Figures 10 and 11)



(a) NETIR (b) VMCPU (c) Covcon

(d) Circle (e) Electricity (f) Covertype

Figure 7. Average Accuracy v/s Inference Latency for all datasets with additional baselines. Error bars correspond to one standard error.

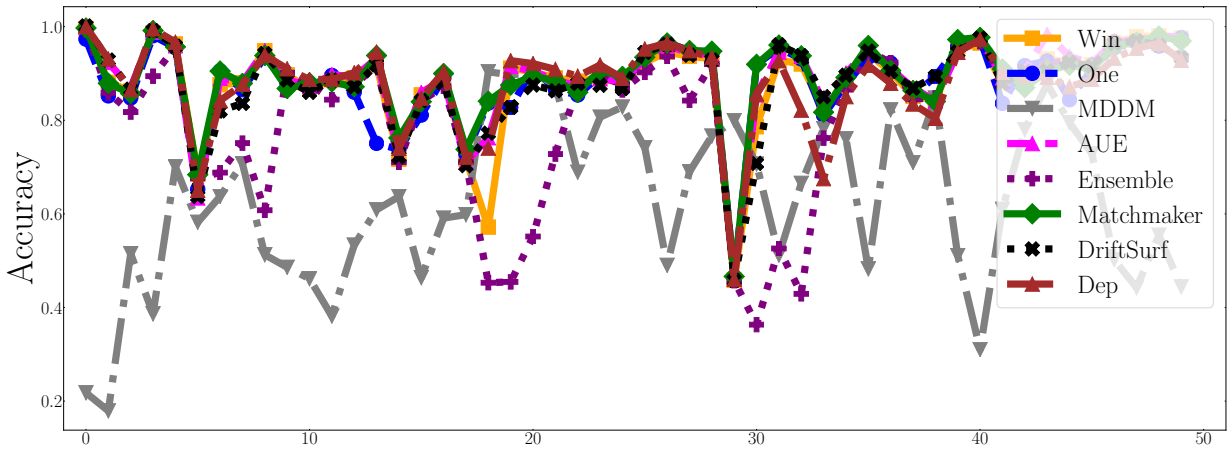


Figure 8. NETIR Accuracy over time

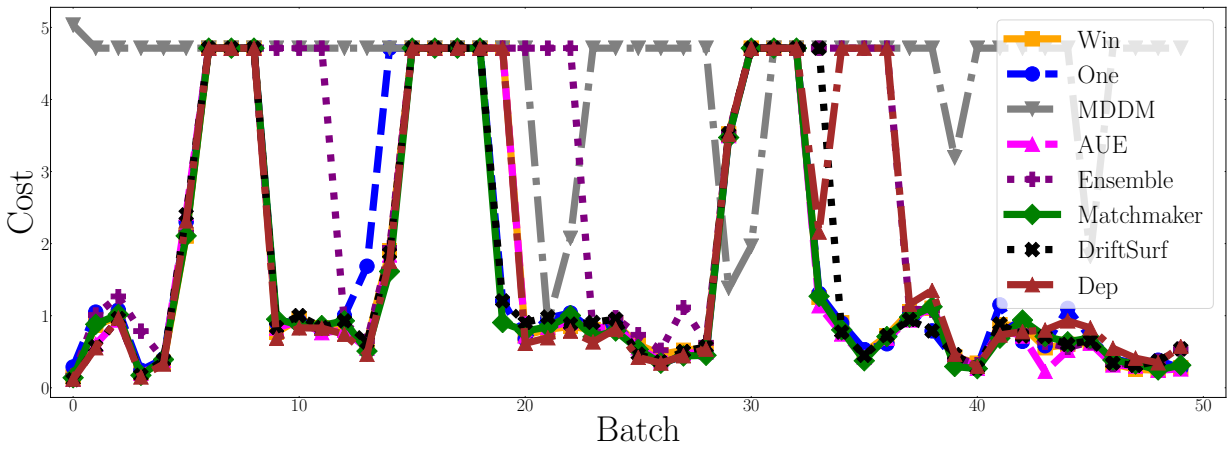


Figure 9. NETIR Cost over time

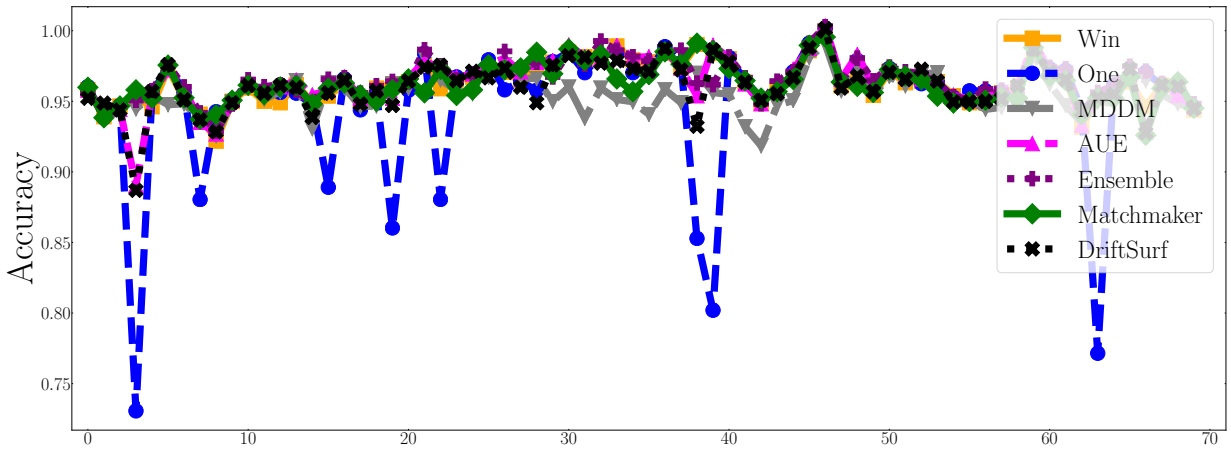


Figure 10. VMCPU Accuracy over time

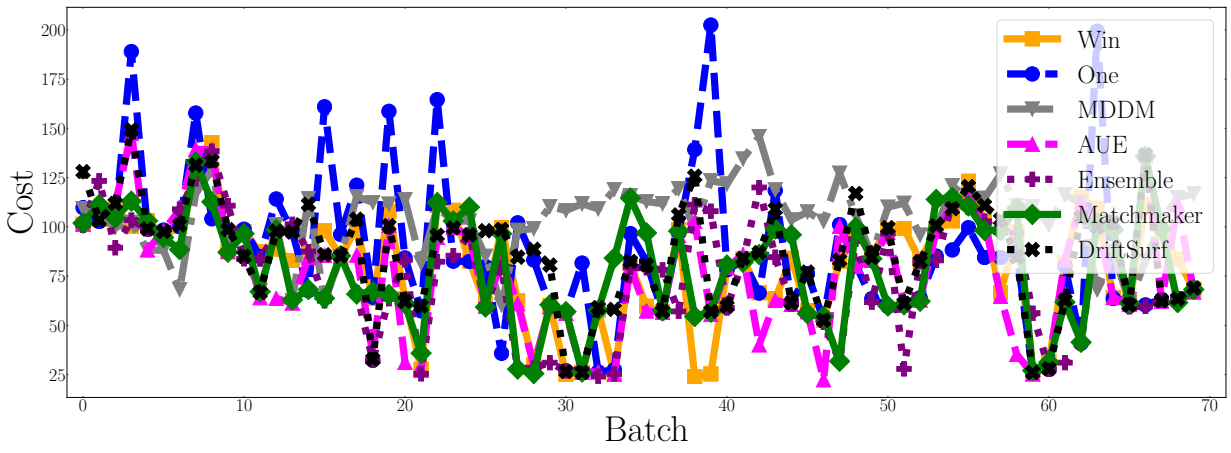


Figure 11. NETIR Cost over time