Nazar: Monitoring and Adapting ML Models on Mobile Devices

Wei Hao
wh2473@columbia.edu
Columbia University
New York, USA

Zixi Wang
zw2774@columbia.edu
Columbia University
New York, USA

Lauren Hong
lauren.hong@columbia.edu
Columbia University
New York, USA

Lingxiao Li
ll3504@columbia.edu
Columbia University
New York, USA

Nader Karayanni
nk3041@columbia.edu
Columbia University
New York, USA

AnMei Dasbach-Prisk*
adasbachprisk@ucsd.edu
University of California San Diego
San Diego, USA

Chengzhi Mao
cm3797@columbia.edu
Columbia University
New York, USA

Junfeng Yang
junfeng@cs.columbia.edu
Columbia University
New York, USA

Asaf Cidon
asaf.cidon@columbia.edu
Columbia University
New York, USA

Abstract

ML models are increasingly run locally on mobile devices for low-latency inference and offline operation. However, it is hard for ML operators to track on-device model accuracy, which can degrade unpredictably (e.g., due to local data drift). We design Nazar, the first end-to-end system for continuously monitoring and adapting models on mobile devices without requiring feedback from users. Our key observation is that accuracy degradation is often due to a specific root cause, which may affect a large group of devices. Once Nazar detects a degradation affecting a large number of devices, it automatically pinpoints the root causes and adapts the model specifically to them. Evaluation on two computer vision datasets shows that Nazar consistently boosts accuracy compared to existing approaches by up to 19.4%.

CCS Concepts: • **Computing methodologies** → Anomaly detection; **Semi-supervised learning settings**; • **General and reference** → **Design**; • **Software and its engineering** → **Software design engineering**.

*Work done at Columbia University.

Keywords: Machine Learning System Design; Root Cause Analysis; Drift Adaptation; Mobile Devices

ACM Reference Format:
Wei Hao, Zixi Wang, Lauren Hong, Lingxiao Li, Nader Karayanni, AnMei Dasbach-Prisk, Chengzhi Mao, Junfeng Yang, and Asaf Cidon. 2025. Nazar: Monitoring and Adapting ML Models on Mobile Devices. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3669940.3707246

1 Introduction

Companies such as Google, Meta, Apple and LinkedIn, are increasingly running their ML models *on users' mobile devices*, so they can run inference locally at a low latency. Examples include text suggestion [9, 49], ranking posts, object detection and tracking for virtual reality [66], speech recognition [43] and targeted advertising [64]. As state-of-the-art ML models are very large, they are typically trained in the datacenters with fleets of GPUs [47, 66]. A team of *ML operators* is usually responsible for retraining the models and pushing new model versions to user mobile devices.

However, in large-scale settings, ML operators lack sufficient visibility into the performance of the models running on users devices [54]. Model accuracy across devices is highly variable over time due to unexpected shifts in the input data (termed *data drift*) or hardware issues in specific devices (e.g., low-quality cameras, microphones) [10, 47]. Although ML operators may fine-tune or retrain models to improve accuracy, it is quite difficult today to detect data drift in the first place or verify whether accuracy has been restored properly after fine-tuning [18, 47, 54–56]. The crux is that after the models are deployed, ML operators have no "ground truth" on how accurate their models are because users do not manually label data.

Many algorithms have been proposed to detect data drift [21, 37, 61], and adapt to drift [34, 38, 50, 57, 67], but they often operate with impractical assumptions (e.g., assume a single source of drift, fully-labeled data, or require significant computational resources), and have never been tested in an end-to-end system.

We present Nazar, the first end-to-end system for continuously monitoring and addressing data drift in large-scale ML deployments on mobile devices. We design Nazar to operate fully automatically without human involvement. It consists of three main functions: (a) *detect* when data drift occurs on individual mobile devices; (b) *analyze* the root cause of the data drift on the cloud; and (c) *adapt* to the cause that triggered the data drift, re-deploying new model versions back to the devices, all without requiring any user input. To enable its practicality for large-scale ML deployments, Nazar only employs *self-supervised* methods, which we describe below.

Detection. Nazar adopts a *confidence threshold* method that detects data drift by comparing the model's confidence of the predictions against a threshold. This method provides good accuracy and is computationally lightweight to run locally on any mobile device.

Root cause analysis. A classical approach [2] for root cause analysis is called *frequent itemset mining*, which finds sets of attributes (in our case, root causes), that are tied with specific data types (detected data drift). Frequent itemset mining can identify root causes only if they are defined as attributes. This technique is popular due to its effectiveness in revealing underlying causes of drift when the data contains a large set of attributes. However, this method tends to generate numerous duplicates or overlapping sets, and therefore is often used in settings in which a human manually inspects the results and acts upon them [1, 4].

Since manual inspection of each cause is untenable in large-scale settings, we introduce novel *set reduction* and *counterfactual analysis* algorithms, two heuristics that automatically produce a small set of likely data drift root causes by eliminating those that subsume each other or overlap.

By-cause adaptation. The vast majority of existing adaptation techniques assume labeled data, and even those that do not, make completely impractical assumptions. They assume a single source of data drift [63, 68], which leads to poor accuracy when there are multiple sources of drift. Instead, we propose a novel *by-cause* model adaptation method, which selectively adapts the model only to the specific cause of accuracy degradation.

The three Nazar components continuously reinforce each other in an end-to-end loop. As our experiments show, Nazar's analysis accurately identifies the root causes of the evaluated drifts, so that by-cause adaption is applied only to data drifted for the same cause. Otherwise, a model adapted to one cause is unlikely to have high accuracy on data that drifted due to completely different causes. Once an adapted model is deployed, Nazar's detection will leverage the new model's confidence to detect new drifts and diagnose them continuously. These components together enable Nazar to effectively obtain a consistently and significantly higher accuracy than existing approaches that adapt on all inputs, without requiring any manual work.

We implement Nazar on Amazon AWS, and evaluate it end-to-end on two datasets: cityscapes, a self-driving car dataset composed of images taken from driving vehicles in various European cities [11], and an ImageNet-derived dataset that emulates a species classification app. We also created several microbenchmark sub-datasets, including one that incorporates traffic objects in rainy scenes [33]. Our results show that Nazar boosts accuracy by an average of 14.9% and up to 19.4% on all data and an average of 31.2% and up to 49.5% on drifted data compared to the baselines on cityscapes. We will open-source our datasets and code upon publication. Our main technical contributions are:

1. First end-to-end online system for model monitoring and adaptation for mobile devices.
2. Fully-automated root cause diagnosis that uses set reduction and counterfactual analysis to narrow down the root causes of model degradation.
3. New self-supervised adaptation technique that adapts by root cause, boosting accuracy significantly over existing approaches when deployed with Nazar's effective root cause analysis, without any human input.

2 Background and Related Work

Companies deploy models on user devices for faster inference and to support low connectivity settings. On-device models support a wide range of use cases, including object recognition [66], text auto-complete [9, 49] and ranking posts and ads [64, 66].

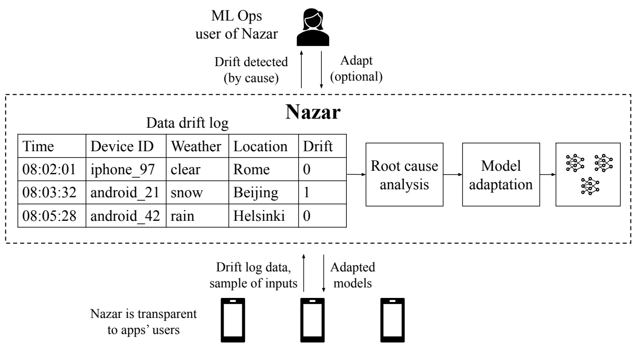
Where to train the models? The models can be trained either centrally in the cloud or in a distributed fashion across the devices themselves (termed *federated learning*). While the latter has seen increased interest in both academia [14, 19, 29, 30] and industry [9, 49, 64] because it does not require uploading training data to the cloud and preserves privacy, cloud is still industry's predominant method because it simplifies the training and evaluation process greatly and enjoys the powerful datacenter AI compute capabilities (e.g., large GPU clusters), even though it offers lower privacy guarantees. We therefore design Nazar to focus on cloud training. Nazar's principles and ideas can also be applied to federated learning.

Accuracy degradation caused by model compression. Some deployment pipelines require models to be adapted to run on resource-constrained devices [15, 59]. However, techniques like model compression can introduce subtle changes in the models, which may degrade their accuracy. These discrepancies in accuracy across devices can impact the applications' quality of service [10, 47], and lead to security vulnerabilities [17, 47]. For example, while quantization can reduce the model size exponentially to fit on resource-constrained devices, it can also lead to worse accuracy for specific classes [10]. This accuracy degradation is hard to anticipate in advance.

Data drift. Data drift is a classic problem which affects models that operate on streaming data, where the distribution of the newly-arriving data diverges from training data's distribution [13, 39, 65], i.e., $p_{test}(x, y) \neq p_{train}(x, y)$. Specifically, there are two common types of drift [48]: (a) covariate drift where $p(x, y) = p(x)p(y|x)$ with $p_{test}(x) \neq p_{train}(x)$ and $p_{test}(y|x) = p_{train}(y|x)$, and (b) label drift where $p(x, y) = p(y)p(x|y)$ with $p_{test}(y) \neq p_{train}(y)$ and $p_{test}(x|y) = p_{train}(x|y)$. These drifts can occur for many reasons: temporal environmental changes or sensor-related issues affecting the model input (e.g., weather, problems with device microphone or camera lense), changes in the input appearance itself (e.g., a self-driving model trained in the US might not recognize street signs in India), and changes in the label distribution (e.g., the number of cars exceeds the number of pedestrians in the deployed environment). The first two are examples of covariate drift and the last one is an example of label drift. To deal with data drift, models are frequently retrained with fresh data, even if some older data is retained, so their training data better reflects the types of inputs they will encounter during inference and thus retain its accuracy [5, 42]. The data drift problem is exacerbated in on-device deployments, because different devices may see divergent data distributions but adapting per device is quite costly. Therefore, an ideal solution should adapt for a set of devices affected by the same drift cause.

Ekya [5] is a recent system that tackles a variant of this problem, by jointly scheduling retraining and inference on the same "edge" servers – servers with multiple GPUs sitting at the network's edge (e.g., in a content distribution network), in contrast to the resource-constrained mobile devices on which Nazar runs inferences. A second key difference is that Ekya's adaptation requires labels so it produces them by running very large models on the edge servers, a luxury that Nazar does not have.

Lack of visibility, monitoring, and automation. Several systems increase visibility and monitoring in ML pipelines focusing on pre-deployment validations. ML-EXray [47] enables ML operators to insert logging instrumentation and debugging assertions. It targets pre-processing issues, quantization bugs, and suboptimal kernels. Another data validation system implemented at Meta [53] identifies possible attributes that caused model performance drop, from the training input. In contrast, Nazar focuses on post-deployment issues caused by data drift.



Time	Device ID	Weather	Location	Drift
08:02:01	iphone_97	clear	Rome	0
08:03:32	android_21	snow	Beijing	1
08:05:28	android_42	rain	Helsinki	0

Figure 1. Nazar's design.

NannyML [41] diagnoses data drifts by associating accuracy degradation with detected drifts on the input. It assumes a single cloud-based model and relies on observing all inference data, and is therefore unsuitable for large-scale mobile deployments. It also lacks automated adaptation.

In summary, while there are existing systems that identify problems in model pre-deployment, and existing ad-hoc detection and adaptation techniques, to the best of our knowledge there is no automated end-to-end system for monitoring and adapting models on mobile devices.

3 Design

3.1 Overview

We design Nazar (Figure 1) to be an end-to-end system that can automatically monitor data drift, identify the main root causes for the drift, and adapt models to them at scale, while dynamically evolving with changing data distributions.

Modes of operation. The user of Nazar is the ML ops team, who maintains and retrains models. ML ops can choose their level of interaction with Nazar: by default, Nazar runs in an "autopilot" mode, where monitoring, analysis, and adaptation are all done automatically. The ML ops team can also manually interact with the system, receiving alerts when data drift occurs, and manually deciding when to trigger the adaptation and on which root causes to apply them.

On-device detection. Nazar is transparent to the users of the apps that run the AI models. Nazar operates in the cloud, and interacts with the on-device models through an API. The apps contribute two types of information to Nazar. First, each time the models conduct inference, they also subsequently run a very lightweight drift detection algorithm (described in §3.2). The result of this algorithm, along with additional metadata attributes about the device and model (e.g., the device's ID, current location, local time, and model version) are sent to Nazar in the cloud. We call this metadata collected from the devices a *drift log entry*. Second, along with this metadata, the device samples a percentage of the actual input data and send it to the cloud for model adaption.

Root cause analysis. The drift log entries are ingested into a database in the cloud, called the *drift log*, which contains only the drift detection results and metadata. Periodically, Nazar runs a *root cause analysis* algorithm (§3.3) that runs (a) a series of database queries on the drift log to identify potential causes – subsets of attribute values – of data drift, and (b) *set reduction* and *counterfactual analysis* algorithms, which reduce redundant causes and rank them to select the most important ones for adaptation. If any cause is found, Nazar initiates adaptation and optionally alerts the ML ops team.

By-cause adaptation. Leveraging the sampled data associated with the drifted entries, Nazar generates new *by-cause model adaptations* that adapt to the specific root causes of the drift (§3.4). These adaptations are pushed to the user devices to serve new inferences. Since model updates can accumulate, Nazar consolidates the different versions to capture the relevant root causes with a small number of model adaptations. For inference, the device chooses which model version to use for each input, by selecting the one with attributes that best match the input metadata.

Evolving drift detection. In the process of detecting, analyzing and adapting to drift, the concept of what is drift and what is not evolves over time. After Nazar adapts on a certain cause of drift and deploys the adapted model, the detection and root cause analysis components automatically calibrate to the new distribution. The three Nazar components thus work seamlessly together to keep up with the continuously drifting distributions.

Design principles. The main question in designing Nazar is how to design its detection, root cause analysis and adaptation mechanisms. Two primary principles guide our design:

1. As Nazar is transparent to app users, it requires *self-supervised* methods that cannot rely on labeled data.
2. Nazar must support fully-automated operation, without relying on the ML ops team instruction.

3.2 On-Device Data Drift Detection

We now describe Nazar's data drift detector, which runs on the mobile devices. The goal of the data drift detector is to detect when data drift has occurred on a particular device in a lightweight fashion, without requiring any user input.

3.2.1 Data Drift Detection Techniques. When designing Nazar's drift detection mechanism, we considered different data drift (also known as "out-of-distribution drift") algorithms from the ML literature. However, most of these techniques were not designed for an on-device inference setting. Table 1 summarizes the eight primary techniques we evaluated. We introduce the classes of techniques, and describe our rationale for focusing on methods that are derived from the model's inference output.

Detecting based on model output. A major class of data drift detection techniques assumes that data drift is correlated with the model being "unsure" about the correct class to output. Therefore, they apply various metrics on the model's *logit* vector (or un-normalized log probabilities). As a refresher, in deep neural network models, the model outputs an array that assigns a probability for each class. Typically the highest scoring class is chosen as the predicted class. The distribution of this vector can indicate how "certain" a model is about its prediction. For example, the MSP *threshold* [21] checks if the maximum softmax value of the logit vector is below a certain threshold. If it is, the model is uncertain about the final prediction, which is correlated with data drift. Prior work also proposes other types of thresholds or metrics, such as the entropy of the softmax values [61] and other similar scores [37]. In our experiments, we found these thresholds to perform almost identically to MSP. MSP has the small advantage that it is normalized between 0 and 1, making it a simpler knob to tune, and hence we adopt it. The advantage of this class of algorithms is that they can simply be applied to the model's inference output with negligible computational overhead (since the logit scores are computed by the inference anyways), and do not require any outside information.

Statistical test on a batch of outputs. Orthogonal to the aforementioned methods that use a simple threshold for data drift detection, statistical tests can be combined with any of those scores (e.g., MSP and entropy) to determine whether the inference data diverges from the training data distribution [61]. For example, Kolmogorov-Smirnov test (KS-test in Table 1) compares the empirical cumulative distribution functions (CDFs) of the two sets of scores and calculates their maximum difference. Prior work [48] conducts empirical studies on using different combinations of statistic test and finds that applying KS-test on softmax outputs yields the best detection accuracy. Hence, this is the main statistical test we evaluate.

External datasets. Another class of detection methods assumes training-time access to a dataset that contains drifted data samples. The intuition is that such a "drift dataset" will help train a model whose logit outputs are more sensitive to all drifts. Examples include Outlier-Exposer [22] (OE in Table 1), Odin [35] and Mahalanobis Distance [31] (MD). We deem these approaches impractical for our setting, because our data is unlabeled, and we cannot assume that users will

	Threshold [21]	KS-test [48]	OE [22]	Odin [35]	MD [31]	SSL [23]	CSI [58]	GOdin [25]
No secondary dataset	✓	✓	✗	✗	✗	✓	✓	✓
No secondary model	✓	✓	✓	✓	✓	✗	✗	✓
No backpropagation	✓	✓	✓	✗	✓	✓	✓	✗
No batching	✓	✗	✓	✓	✓	✓	✓	✓

Table 1. Comparison of different data drift detection algorithms from the ML literature. Nazar uses the threshold method, which simply applies a threshold on the model's softmax output, such as the max softmax score.

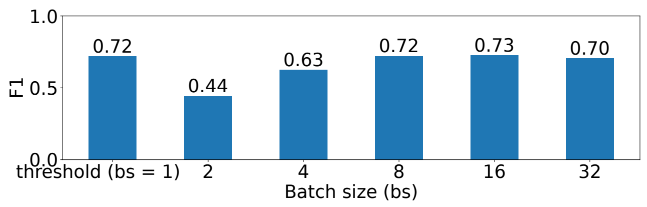


Figure 2. Comparison of F1 scores using KS-test with different batch sizes. For batch size is 1, we use threshold on MSP with default value 0.9.

prepare drift datasets which is a meaningless concept for a user.

Secondary model. Some techniques employ auxiliary models to detect data drift. SSL [23] and CSI [58] co-train an auxiliary model for self-supervised tasks on a transformed version of the dataset (e.g., they rotate images in the original dataset and perform rotation angle classification). The assumption is that if there was no drift, after training, both models would have high confidence on the same input. We rule out this class of methods, given that some mobile devices are resource-constrained, and we cannot rely on them to invoke an additional model purely for data drift detection.

Methods that require backpropagation. Generalized Odin [25] (GOdin) is an expanded version of Odin that does not require the use of a secondary dataset but requires adding small adversarial perturbations to make the model more confident when data is in distribution, and less confident when data has drifted. However, to add these perturbations, it requires an extra step of *backpropagation* after the softmax values are read, i.e., the neural network needs to be traversed in reverse, then followed by another step of inference on the perturbed input. Unfortunately, this method triples the inference time which makes it unsuitable for Nazar's lightweight detection design.

3.2.2 Chosen Detection Technique: Threshold on MSP.

After ruling out most of the techniques due to their lack of fit with on-device inference, we finalize a shortlist of two detection methods: the simple MSP threshold method that operates on one model-outputted logit vector at a time, and the KS-test statistical method, which operates on a batch of MSP scores. To evaluate these methods we measure their *F1 score* to compare their effectiveness on an equal split of clean and drifted images (described in §5.3) and a ResNet50 model (see §5.2) is used to generate the logit vectors. The F1 score is defined as:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = 2 \cdot \frac{TP}{2TP + FP + FN} \quad (1)$$

where a high score indicates that the detector has a good balance between precision and recall. In other words, the model is able to correctly identify true positives (the image is "drifted", i.e., it belongs to a different distribution than the training set, and the detector thinks it is drifted) while minimizing false positives (the image is not drifted and the detector thinks it is drifted) and false negatives (the inverse). For the KS-test method, we test it with different batch sizes and assign the detection result (i.e., a boolean value for drift or non-drift) *on the whole batch*. For the threshold method, we set its default value to 0.9 (we evaluate this choice in §5.3).

Figure 2 displays the results, showing that when the batch size is higher than 4, KS-test slightly outperforms the threshold method, but is worse when the batch size is lower. However, batching results from device inference raises various tricky questions: should one batch inference results from a specific device over time? How long should we allow that time window to be? How do we batch results when we do not have sufficient samples from a single device to fit in one batch?

Since the threshold method performs very similarly to KS-test with large batches, and since using a batched detection method raises these thorny issues, we choose the threshold method with MSP as the default detection module in Nazar.

3.3 Root Cause Drift Analysis

As we will show later (§5), since the drift detection algorithm operates without any user input or labeled data, and it is simply a function of the model's uncertainty, the detection algorithm is somewhat noisy for each individual detection. Therefore, we need a more accurate mechanism at a system level to determine whether drift has actually occurred, and why. This subsection explores the design of this mechanism, which we call the *root cause drift analysis*.

In Nazar, for each inference, the device sends to the cloud the data drift detection results, as well as metadata about the device and its environment (e.g., weather and geolocation, as depicted by the example in Table 2). Each entry is called the *drift log entry*, and the entries are assembled in a single

Time	Device ID	Weather	Location	Drift
06:02:01	android_42	clear-day	Helsinki	False
06:02:23	android_21	clear-day	New York	False
06:04:55	android_21	clear-day	New York	True
08:03:32	android_21	snow	New York	True
11:05:01	android_42	snow	Helsinki	True

Table 2. Example of drift log.

large database in the cloud, called the *drift log*. Hence, the drift log is a live global view of data drift across all devices.

We walk through the algorithm using an example of two devices (one in New York, one in Helsinki) running an application for classifying animals in their natural settings. This application produces a drift log (Table 2). Each device generates a few drift entries, which contain metadata on the current location of the device and the weather it is operating in. This metadata can be either gleaned by the device itself (e.g., its location) or generated by Nazar in the cloud via an external source (e.g., for weather metadata, Nazar looks up the weather in a third-party weather API using the device's location). In this example, the root cause of the drift is the snowy weather, which transforms the image sufficiently so that it diverges from the original training data – images taken in clear days – of the model. The example includes a false positive drift detection, i.e., the third entry in the log, where the detection threshold algorithm marked the image as "drifted" (due to low confidence logit scores), even though the image was not.

Frequent itemset mining (FIM). To identify whether drifts exist and discover its root causes in the face of potential false positives and negatives, Nazar focuses only on clusters of drifts that are statistically significant. One possible approach is to directly apply an ML-based clustering algorithm on the entries flagged as drifts. We did not adopt this approach because clustering algorithms often require specifying how many clusters to divide the data into (e.g., the K parameter in K-means), and can be expensive, especially at scale [52].

An alternative, classical data mining approach [2] is to explore whether sets of attributes in the drift log entries, e.g., {snow, New York} in Table 2, frequently appear with the drift attribute. These attributes are termed in the data mining literature as *frequent itemsets*, and there exist many algorithms [3, 7, 8, 16, 44, 45] for frequent itemsets mining (FIM).

In the first stage of root cause analysis, shown in Figure 3(a), Nazar implements an FIM mechanism that employs the *apriori algorithm* [4], a common method for FIM. It first identifies frequent individual attributes in the entry and then uses them to generate larger sets of attributes associated with the drift attribute. The algorithm calculates several metrics starting from sets containing each single attribute, and then

Rank	Metrics				Attributes		
	Occ	Sup	RR	Conf	Weather	Location	Device ID
0	0.4	0.67	3	1	snow	-	-
1	0.2	0.3	2	1	snow	-	android_21
2	0.2	0.3	2	1	snow	-	android_42
3	0.2	0.3	2	1	snow	New York	-
4	0.2	0.3	2	1	snow	Helsinki	-
5	0.4	0.7	1.3	0.67	-	-	android_21
6	0.4	0.7	1.3	0.67	-	New York	-
7	0.4	0.7	1.3	0.67	-	New York	android_21
8	0.2	0.33	0.75	0.5	-	-	android_21
9	0.2	0.33	0.75	0.5	-	-	android_42
10	0.2	0.33	0.75	0.5	-	Helsinki	-
11	0.2	0.33	0.75	0.5	clear-day	-	android_21
12	0.2	0.33	0.75	0.5	-	Helsinki	android_42
13	0.2	0.33	0.75	0.5	clear-day	New York	-
14	0.2	0.33	0.75	0.5	-	Helsinki	-
15	0.2	0.33	0.33	0.33	clear-day	-	-

Table 3. Example of frequent itemset mining results where Occurrence, Support, Risk Ratio and Confidence are abbreviated as Occ, Sup, RR and Conf.

the sets containing combinations of attributes which form subsets of the single ones, e.g., entries with {snow, New York} are a subset of entries with {snow}. (For clarity, henceforth we refer to the entries with attribute values {v1, v2, ...} simply using {v1, v2, ...}.) The algorithm then filters the sets by checking if statistics from the metrics pass certain thresholds to consider a certain set as a cause of drift, since the drift detection is noisy. Finally, it ranks which sets of attributes are the leading causes of drift. The metrics and ranking for the example log in Table 2 are shown in Table 3.

The *occurrence* of a set of attributes measures how often it appears in the drift log. The *support* of a set of attributes measures how often it is marked as "drift" as a proportion of all log entries that are marked as drifted. E.g., {snow} has a support of 0.67, because $\frac{2}{3}$ of the drift entries have the attribute "snow". The *confidence* of an attribute set is how often it is marked as drift as a proportion of all its appearances in the table. Finally, the *risk ratio* of a set measures how much likelier entries that are marked as drifted contain the set of attributes versus not containing it. E.g., for {snow, Helsinki}, the risk ratio is 2, because the probability of an entry being drift given it contains {snow, Helsinki} is 1 and the probability of an entry being drift given it does not contain {snow, Helsinki} is $\frac{1}{2}$.

By default, Nazar uses the risk ratio to rank the results, because it measures the importance of a specific root cause. Following prior work [1], Nazar sets the maximum number of attributes that can belong to a single root cause to 3, and uses values of 0.01, 0.01, 0.51 and 1.1 for minimum occurrence, support, confidence and risk ratio respectively, by default.

Practical limitations of FIM. While FIM is a useful technique to rank root causes and set some thresholds on which

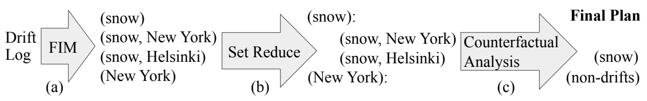


Figure 3. Flow of root cause analysis.

root causes are important, systems that rely on it typically assume that a human will then inspect the results and manually choose the appropriate root causes [1]. In our example in Table 3, the top seven rows are all possible root causes, since they pass the four thresholds. However, it would not be a good idea to simply adapt models to each one of these root causes, since they are overlapping and even contain subsets of each other (e.g., {snow, New York} is a subset of {snow}). Therefore, Nazar needs a way to narrow down this set of possible root causes so it can apply model adaptations sparingly, covering as many relevant root causes as possible without requiring operator instruction.

Set reduction. There are two types of possibly redundant root causes that may have already been addressed in other root causes, and thus can be *reduced*. The first type is the root cause which is a subset of other root causes that have broader coverage. For example, {snow, New York} is a subset of both {snow} and {New York}, so if one of these two is selected as root causes for model adaptations, there is no reason to select their intersection,e.g., {snow, New York}. Nazar's set reduction algorithm merges the subset root cause into one of its super-sets that has higher ranking, e.g., {snow, New York} is merged into {snow} instead of {New York}, because {snow} is ranked higher (rank 0). This stage is termed as *set reduction* and is depicted in Figure 3(b).

Counterfactual analysis. The second cause of redundancy is overlapping causes, when the entries with drift are already "covered" by some other root causes. In our example, $\frac{2}{3}$ entries from New York contain drift signals while $\frac{1}{3}$ are false positives. However, this subset of images is already addressed by a higher-ranked root cause, e.g., {snow}. Nazar filters out this type of root cause by applying counterfactual analysis [32], which iteratively modifies the drift attribute from the log entries associated with a higher-ranked root cause to be "false" and tests whether the lower-ranked root cause is still statistically significant by checking the four metrics after these drift attributes have been modified. If it is still statistically significant, Nazar adds this lower-ranked root cause into the final result. Note that the first top-ranked root cause is directly added to the final result. This iterative process is shown in Figure 3(c) and it runs until all root causes from the set reduction output are exhausted. Eventually, Nazar groups images associated with each filtered root cause, e.g., images that have "snow" in their meta-data. Note that Nazar also filters a set of images that are "clean" when they are not associated with previously discovered root causes. Algorithm 1 describes the root cause analysis algorithm.

Limitations. The primary limitation of our root cause analysis is that the drift log attributes may not capture all possible root causes of drifts. For example, certain devices may have camera lenses from a particular manufacturer causing image corruption, but the drift log might not contain an attribute of the lens manufacturer for each device. In such cases, Nazar may group devices by their models, their location, or even by device ID, even though the "real" root cause is the lens. It should still automatically produce by-cause adaptations, which would capture these data drifts, but the ML operator team would have to manually investigate the issue in order to understand the real root cause. To combat this issue, in large-scale settings, the list of attributes should be as exhaustive as possible, to capture unforeseen causes of drift.

Algorithm 1 Root cause analysis algorithms

```
FIM() :
Output: Extracts frequent sets of attributes associated with drift.

Set_Reduction(FIM_table) :
Input: Sorted list of attribute sets.
Output: Mapping between the most coarse-grained attribute sets and their
subsets. Ties between coarse-grained sets are broken by ranking.

Passes_Drift_Threshold(items) :
Input: Attribute set potentially causing drift.
Output: True if the set passes thresholds (occurrence, support, confidence,
and risk). False otherwise.

Drift Analysis:
1: FIM_list ← FIM()
2: coarse_associations ← Set_Reduction(FIM_list)
3: root_causes ← [ ]
4: while coarse_associations do
5:     curr_coarse_cause ← coarse_associations.pop()
6:     if Passes_Drift_Threshold(curr_coarse_cause) then
7:         root_causes.append(curr_coarse_cause.key)
8:         Mark_No_Drift(curr_coarse_cause.key)
9:     else
10:        for each subset in curr_coarse_cause.value do
11:            if Passes_Drift_Threshold(subset) then
12:                root_causes.append(subset)
13: return root_causes
```

3.4 By-Cause Adaptation

Most model adaptation techniques [34, 38, 50, 57, 67] and end-to-end ML systems that incorporate adaptation (e.g., SageMaker [36], TFX [40]) require labeled data, which cannot be assumed in our setting. We focus on self-supervised methods for adapting models to data drift, motivate the idea of adapting a model to a *specific cause of drift*, and introduce how Nazar chooses different adapted model versions for inference.

Self-supervised adaptation. Nazar's model adaptation builds on prior works on self-supervised adaptation [63, 68]. The intuition is to select a self-supervised objective (e.g., minimize the entropy on the logits output by the model) that captures some invariant of the data itself without requiring any labels. Upon data drift, this objective is no longer optimal, therefore we can optimize the model weights to minimize this objective again, effectively adapting the model to the drift. Specifically, Nazar minimizes the model prediction entropy (TENT [63]):

$$L(\theta; x) = -\sum_{c} p_\theta(\hat{y}_c|x) \log p_\theta(\hat{y}_c|x) \tag{2}$$

where θ denotes parameters of the model, $\hat{y}_c$ is the model's predicted probability of class c. Note that optimizing a single prediction has a trivial solution: assign probability= 1 to the most probable class. Nazar prevents this by jointly optimizing batched predictions over parameters that are shared across the batch. Moreover, since adapting a whole model including all its layers significantly increases network consumption and model storage costs because each adaptation leads to a whole new version of the model weights. Nazar adapts only the *batch normalization (BN) layer* [27] for efficiency. As an example, in ResNet50 the BN layer is 217× smaller than the full model (0.4MB vs. 92MB). Note that ML operators are free to employ other objectives. For instance, another common objective is minimizing marginal entropy (MEMO [68]), which is the entropy of the averaged output logits over a set of randomly augmented copies, $\tilde{x}_1, ..., \tilde{x}_B$, of a single image, e.g., by rotating and posterizing the image:

$$L(\theta; x) = -\frac{1}{B}\sum_{i=1}^{B}\sum_{c} p_\theta(\hat{y}_c|\tilde{x_i}) \log p_\theta(\hat{y}_c|\tilde{x_i}) \tag{3}$$

This method allows adaptation on a single input by introducing random augmentations but also incurs too frequent adaptations even for one-off or falsely-detected drifts, which is impractical in our setting. And thus we adopt it using the setups similar to TENT, where it adapts only the batch normalization layers of a model based on a small batch of inputs.

Benefit of adapting by cause over adapting blindly. Prior adaption methods unrealistically assume a single cause behind the data drift and adapt on all collected inputs, but in practice data drift can originate from different sources: for example, a traveler may bring her device from location to location over a short period of time, experiencing very different input distributions each corresponding to a different cause (location). We show in our experiments below that adapting by cause improves model performance significantly over adapting blindly on all inputs on both training objectives. Moreover, to illustrate the need for accurate root cause analysis, we show that a model adapted to one cause of data drift has poor performance when applied to data that is drifted due to a different cause or non-drifted (i.e., clean) data.

Methods	Average Accuracy (%)
No-adapt	38.7
By-cause (TENT)	61.5
By-cause (MEMO)	42.3
Adapt-all (TENT)	42.4
Adapt-all (MEMO)	30.3

Table 4. TENT and MEMO when adapting them by-cause, and using a single model to adapt to all sources.

We adapt a pre-trained ResNet50 on a dataset that contains 16 types of data drifts, such as weather-based corruptions and image blur (§5 describes the evaluation setup), and a portion of clean data. We run adaptation in two settings: (a) adapt and test a model for each cause of drift and clean data (17 total models), termed *by-cause*, and (b) adapt one model on a mix of all 16 sources of drifts and clean data, and test them on the same test set as in (a), termed *adapt-all*. We also evaluate non-adapted ResNet50 on the same test set.

The average test accuracy is shown in Table 4. Adapting on data by-cause using both methods improves accuracy over the original model, by 22.8% for TENT and 3.6% for MEMO; whereas adapting on all the data degrades accuracy with MEMO and leads to small gains with TENT. The reason is that adapting models to multiple divergent distributions without supervision can cause models to underfit and lead to low performance. To further demonstrate this, we run an experiment where we adapt a model by-cause (foggy weather), and test it on images with other sources of drift. The model obtains an average accuracy of only 16.4% when run on images with other sources of drift, and an accuracy of 66.7% on its own test set. Its accuracy on clean data is also very low: 26.8%, while the model adapted on clean data has an accuracy of 74.6% on the same clean test set.

Therefore, we conclude that Nazar has to adapt its models sparingly and apply the adaptations carefully; it should apply adapted models only when there is a true prevalent source of data drift that affects some portion of user devices, and ideally run the resulting adapted model on data that experiences the same source of drift. These observations motivate Nazar to adapt different models by distinct root cause, so that the distribution of inference data will be as close as possible to the one of the training data. Ideally, each by-cause adapted model should run inference on data that belongs to the same distribution it was adapted on, while a continuously adapted "clean" model should be run on clean data. Since no ground truth exists, Nazar has to rely on its root cause analysis to determine which devices are experiencing data drift and from which source. In addition, we decide to use TENT as the default method to adapt the models in Nazar, since TENT largely outperforms MEMO in the both strategies in our experiment.

Consolidating model versions. Adapting by-cause generates multiple model versions. Though Nazar only updates and deploys BN layers, We cannot allow the number of models to grow unbounded, especially on resource-constrained devices. Nazar constrains the number of models that can be stored on each user device, by periodically consolidating them. To consolidate different versions, Nazar keeps a global view of the model pool, in a least recently updated (LRU) list, which contains the sets of models to be deployed. The oldest models make room for the newer ones. Orthogonally to the LRU algorithm, if a new model version belongs to the exact same set of attributes as an existing one, the older one, instead of the tail of the LRU, is evicted. In addition, if an incoming model version has a root cause that is a superset of an older model version, the older version gets evicted (similar to the set reduction algorithm in §3.3).

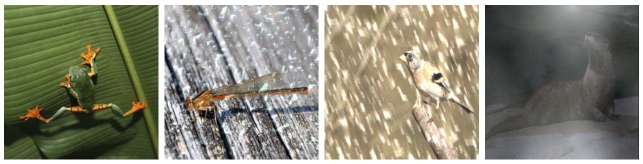
Picking which version to use for inference. During inference on the device, Nazar will use the most-recently updated model that has the highest number of matching attributes for inference, e.g., {rain, New York} has more attributes matching than {rain} if the input image is associated with {rain, New York}. It uses the risk ratio ranking of a root cause, described in §3.3, to break ties between model versions. If no matching model is found, it uses its "clean" model for inference. Note that model selection for inference is run on the device without any involvement from the cloud.

4 Implementation

We now describe our prototype implementation of Nazar on Amazon AWS. To support a large number of devices and models, our implementation uses highly-scalable AWS services. Such services exist on all major clouds, and our implementation could be easily ported to other clouds.

Drift log. We run the drift log on Amazon Aurora [62], a relational database that supports over 120,000 writes per second and can be scaled to 128 tebibytes. The drift log is stored as a table, where each row contains the metadata and if the inference was detected as a data drift.

Root cause analysis and adaptations. The root cause analysis is run periodically as an Amazon Lambda function, which is triggered either automatically based on a configurable time window or manually by the ML operator. The FIM algorithm is run as a set of SQL queries on the drift log. The initial phase of the algorithm requires extracting the attributes frequently associated with drift. This can be implemented using a simple SQL Count aggregation, with appropriate conditions. The function then calculates the FIM metrics, which are used to filter and rank the root causes. The function then runs the set reduction and counterfactual analysis. Finally, each narrowed-down root cause is sent to a GPU-equipped instance, which conducts the adaptation by-cause.


(a) Wildlife animal example.

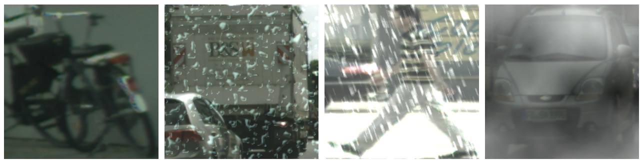

(b) Cityscapes example.

Figure 4. Examples of clean images and images with rain, snow, and fog (corruption severity of 3) from two datasets.

5 Evaluation

This section answers the following questions:

Q1: How well does Nazar's detection algorithm detect different types of data drift, under both synthesized and real weather conditions? (§5.3)

Q2: How effective is Nazar's root cause analysis in identifying the root cause of drift? (§5.4)

Q3: How does by-cause adaptation perform compared to prior approaches with various drift sources? (§5.5)

Q4: Does the detector evolve with model adaptation? (§5.6)

Q5: What is the accuracy in end-to-end workloads? (§5.7)

Q6: How quickly does Nazar adapt to data drift and how well does the root cause analysis scale? (§5.8)

We first describe our datasets (§5.1) and our experimental setup (§5.2), and then answer the evaluation questions.

5.1 Datasets

We build two datasets for our evaluation, Cityscapes and Animals, representing the typical Nazar applications. We emulate both datasets from January 1, 2020 to April 21, 2020 and apply synthetic weather-based data drifts (rain, snow, and fog) [20] based on historical weather information [28, 60] which exhibits diverse weather patterns. By default, we evenly divide the time for both datasets by 8, and run Nazar after each interval. The drifts are applied as different types of random corruptions to the images and whose severity level can be parameterized. For example, if an image from Hamburg is tagged to have been generated on January 18th, 2020, and the historical weather for that day indicated there was rain at some point during that day, by default, we apply a drift function for rain (e.g., see Figure 4) on that image.

Cityscapes. A potential application of Nazar is self-driving cars, which conduct on-device object classification and image segmentation [6]. Similar to prior work (Ekya [5]), we prepare an object detection dataset for a self-driving car application, built on top of the cityscapes dataset, which contains photos of traffic-related objects collected from driving cars from 50 cities in Europe [11] tagged with sequence numbers indicating the temporal order of photo collections. Following Ekya's methodology, we preprocess cityscapes for object classification, resulting in 27,604 images, and use 14% of the dataset for the initial training, 6% for validation and 80% for streaming new inferences. We assume the dataset at each location starts on January 1st, 2020 and ends on April 21st, 2020, and that the images are submitted for inference in equal intervals across these dates.

Animals. In addition to cityscapes, the temporally-tagged dataset, we also generate a parameterized synthetic dataset that would allow us to methodologically configure our system. We emulate a geo-distributed deployment of an animal identifier app, inspired by iNaturalist [26], where subscribers from different locations take photos to identify animal species. We build this dataset with a subset of ImageNet [51], that contains 201 classes of wild life animals. Each of these classes contains on average 793 images for model training, 50 images for validation, and 500 images for streaming. We did not use the original iNaturist [24] because both the data drift functions and adaptation methods were designed and tested for ImageNet [20, 63, 68] and we want to conduct a fair comparison between Nazar and prior techniques.

We emulate 7 locations from different continents: New York, Tibet, Beijing, New South Wales, United Kingdom and Quebec, where each location contains a different distribution of animal species and a configurable number of user devices with a configurable arrival pattern of inference requests. We set a default of 16 devices per location and an arrival Poisson distribution with a mean of two images per day per device. We assume users suffer from weather-related distribution shifts, and apply the same methodology as the cityscapes dataset.

Class skew. We introduce another source of data drift in Animals dataset, *class skew*. We observe that a model's accuracy of images from different classes varies from 39.2% to 98.2% as shown in Figure5b, despite similar the number of training images in each class. Therefore, if a location exhibits a higher proportion of images from lower-accuracy classes, the model's accuracy may suffer. To emulate this effect, we employ a Zipf distribution to assign probabilities to classes in each location, where a higher value of parameter α means a higher degree of skew. By default, we set α to be 0 (uniform) but also evaluate under a high skew ($\alpha = 1$).

Real Rainy Images. Lastly, in order to test how the detection component works on real-world data drift, We compiled a sub-dataset of images taken in rainy conditions, by randomly selecting 19,466 images from the Cityscapes and the RID (Rain in Driving with Objects Label) dataset [33], where half of these images were sourced from the RID. We focused exclusively on the five classes common to both datasets.

5.2 Evaluation Setup

AWS setup. Our AWS setup uses the following configurations. The drift log is run on Amazon Aurora with a 5.7.mysql_aurora.2.11.1 engine on a db.r6g.2xlarge instance. The drift analysis is run as an AWS Lambda function with 256MB of memory. The adaptation is run on a p3.2xlarge EC2 instance equipped with an NVIDIA Tesla V100 GPU.

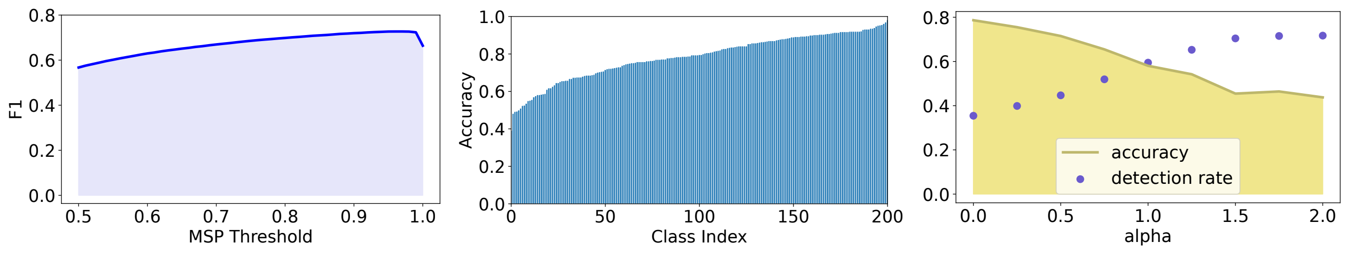
Models. We use three image classification models in all experiments, ResNet18, ResNet34 and ResNet50, which are commonly-used model architectures for mobile devices [46]. Each model is trained from scratch until convergence. The accuracy of each model on clean (non-drifted) validation dataset with cityscapes is 83.6%, 83.9% and 83.7%, respectively, while the accuracy of the base model on the animal dataset is 72.1%, 75.4% and 76.1%.

Data drifts. Prior work on drifts applied 16 types of different data drifts on ImageNet [20]. In microbenchmarks in §5.3, §5.5, and §5.6 we show how Nazar handles all 16 types of drifts. Our end-to-end evaluation (§5.7) contains only the three weather-related drifts (snow, rain, fog), as the distribution of weather is dynamic and driven by real historical weather records. 29% and 36% of days in the cityscape and animal datasets, respectively, experience weather-related drifts. All 16 drifts use a severity level of 3 (out of 5) by default.

Baselines. We compare Nazar with two other settings: adapt-all, which is the baseline approach (e.g., employed by Ekya and previous self-supervised adaptation methods), where one model is continuously adapted on all input during each adaptation window, and no-adapt, which is a pretrained model that is never adapted. Note that besides adapt-all, we also conducted experiments on Nazar to adapt on only the images whose drift attribute is "True," but this method always yields worse performance than adapt-all and thus we do not present its results here.

5.3 Detection (Q1)

To show how well the MSP threshold detects drifts, we measure the F1 scores calculated with different MSP thresholds on the logit outputs from Resnet50. We evenly applied 16 types of drifts on half of the streaming images in the animal dataset, as the positive set, and leave another half non-drifted, as the negative set. We observe that the F1 score of MSP detector steadily increases until it reaches 0.73 and then decreases as shown in Figure5a. The detector's sensitivity to threshold changes is small around a threshold of 0.9, which we adopt as a default value.



(a) F1 score by MSP threshold. **(b)** Accuracy across classes. **(c)** Rate of detection and accuracy vs skew.

Figure 5. 5a shows F1 scores of MSP detector are fairly stable on different threshold values. 5b shows average accuracy on different animal classes is highly variable. 5c shows the accuracy decreases and detection rate increases when skew is high.

Analysis Methods/ Ground Truth Drifts	None	Rain	Snow	Fog	Fog & Snow	Fog & Rain	Snow & Rain	Snow, Rain & Fog
FIM	1	0.919	0.773	1	0.891	0.921	0.926	0.917
FIM with Set Reduction	1	0.919	0.773	1	0.891	0.921	0.934	0.919
FIM with Set Reduction and CF Analysis	1	1	0.874	1	1	1	1	1

Table 5. Evaluation of root cause analysis algorithms with Fowlkes-Mallows Score (1 is optimal).

Further illustrating this point, in Figure 5c, we vary the class skew by increasing α from 0 to 2, i.e., we vary the distribution we sample from different classes, where under high skews we are much likelier to sample specific animal classes than others. The result shows that the detector rate significantly increases from 0.35 to 0.72 and the total accuracy degrades from 78.7% to 43.8% under high class skew.

Detection under real weather conditions. We also tested the detection of real images taken under rain. Our ResNet50 model's accuracy is decreased from 85.2% to 76.7% when switching from the Cityscapes data to the RID data, indicating significant data drift. Notably, our detector achieved a peak F1 score of 0.67 at a threshold of 0.95, with a precision of 0.55 and a recall of 0.88. Although the detector's results are noisier, it remains useful for detecting drift due to rain. We anticipate improved detector performance if the model is trained on additional clean images from the camera that produced the RID dataset, which could help calibrate it to better handle "non-drift" scenarios.

These results together shows that the MSP threshold is a good indicator of different sources of drifts.

5.4 Root Cause Analysis (Q2)

We evaluate Nazar's root cause analysis using the Fowlkes-Mallows Score (FMS) [12], which measures the similarity between two sets of clustering results. In our case, the two sets are ground truth root causes of drift, and ones that are identified by our root cause analysis. The FMS score is:

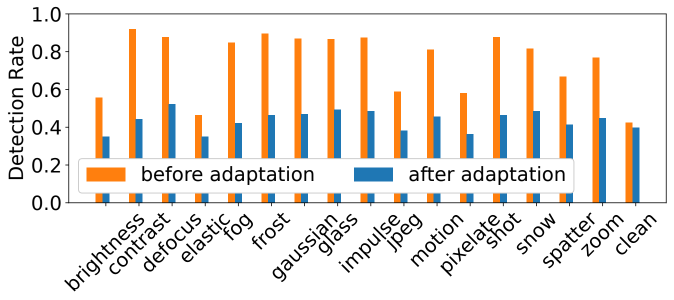
$$FMS = \sqrt{\frac{TP}{TP+FP} \cdot \frac{TP}{TP+FN}} \tag{4}$$

where TP (true positive) is the number of pairs of data points clustered together in both sets, FP (false positive) is the number of pairs clustered together in the second set but not in the first, and FN (false negative) is the inverse of FP. The score ranges from 0 to 1, with higher values indicating more similarity between the clusters. We use 8 drift scenarios, depicted in Table 5, whose ground truth root causes of drifts are different combinations of three weather corruptions on the animal dataset for 14 days starting on January 1st, 2020, on ResNet50. For instance, if the ground truth root causes are "fog" and "snow", we apply only these two types of drifts on the dates that have foggy and snowy weather for each location but ignore "rain" drift on the rainy days.

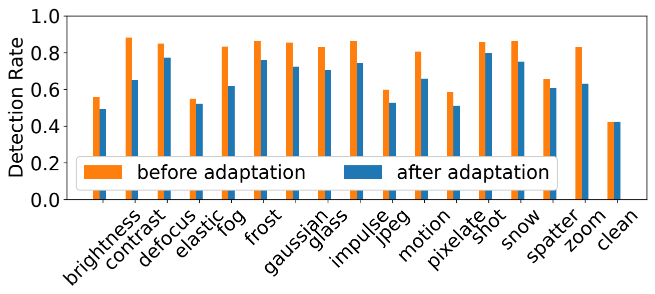
The combination of FIM, set reduction, and counterfactual analysis always yields the highest score under each scenario. In fact, it is optimal under all scenarios except "snow".

5.5 Adaptation (Q3)

To evaluate whether by-cause adaptation performs better than the naive approach, i.e., adapting on all coming images, we evenly split the images in the animal dataset in 17 partitions, 16 of which are transformed with different drifts, and one is left clean, and run them on ResNet50. To isolate the performance of the adaptation mechanism, we assume perfect knowledge of the underlying root cause of the data drift, and that we adapt only to images with that root cause (i.e., we assume Nazar's detector and root cause analysis are perfect). We treat the clean data as its own data source, and run an adaptation model for it as well. We test the adaptation in two settings: (a) when the severity levels of drift, between the images that the model are adapted on and the held-out test images, are both 3 by default, and (b) when the held-out images have a different corruption distribution, i.e., the

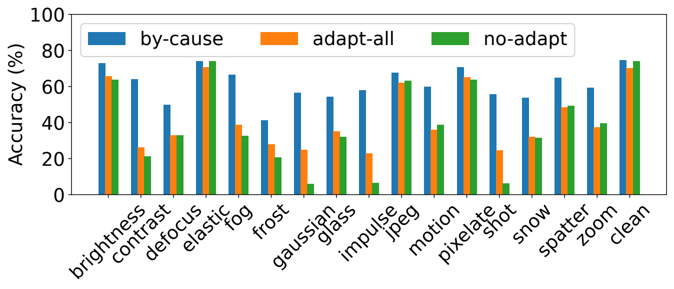


(a) Detection rate of the detector ($S_{adapt} = S_{test}$)

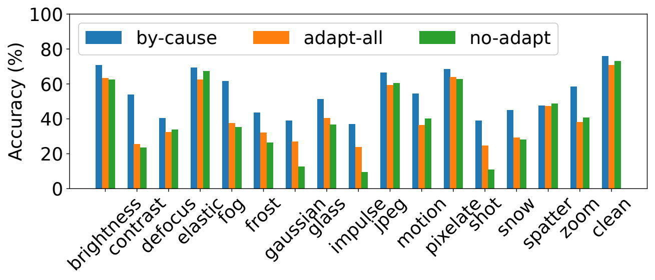


(b) Detection rate of the detector ($S_{adapt} \neq S_{test}$)

Figure 6. Detection rate comparison before and after adaptation. 6a plots the detection rate when the corruption severity levels are identical, while 6b plots it when the corruption severity levels are different, comparing images that the models are adapted on and the test images.



(a) Accuracy of adaptation methods ($S_{adapt} = S_{test}$).



(b) Accuracy of adaptation methods ($S_{adapt} \neq S_{test}$).

Figure 7. Performance comparison of adaptation methods on different drift causes and clean data. 7a plots the accuracy when the corruption severity levels are identical, while 7b plots it when the corruption severity levels are different, comparing images that the models are adapted on and the test images.

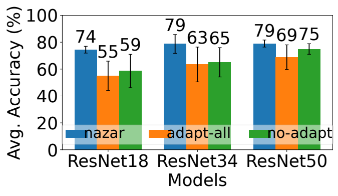
severity level for each drifted image is drawn from a normal distribution with mean=3 and std=1, with rounding and clipping so the result will be an integer from 0 to 5, where 0 means no corruption.

Figure 7a and 7b breaks down the average accuracy by drift type. By-cause adaptation significantly and consistently outperforms the adapt-all. In addition, it provides a significant improvement compared to the baseline non-adapted model, while adapt-all sometimes degrades the accuracy of the non-adapted model. For setting (a), the total average accuracy increase of by-cause adaptation in this experiment is very significant: it achieves a 61.5% accuracy, compared to only 42.4% for adapt-all and 38.7% for the non-adapted model. For setting (b), by-cause adaptation achieves a 54.3% accuracy, compared to only 42.0% for adapt-all and 39.6% for the non-adapted model, which shows its robustness under this more challenging scenario.

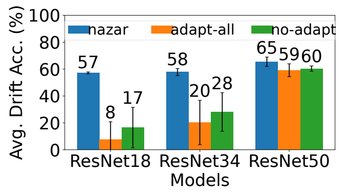
We conclude that by-cause adaptation shows promise in overcoming the limitations of traditional supervised adaptation approaches, paving the way for more scalable and efficient adaptation in real-world applications.

5.6 Evolving Drift Detection (Q4)

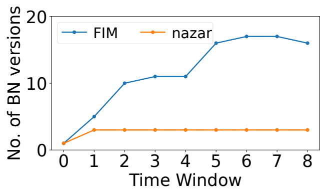
We test how the by-cause adaptation influences the drift detector. Using the same experimental setup as §5.5, Figure 6a and 6b displays the detection rate for each one of the drift types before and after adaptation. The post-adaptation detection rate is measured for the adapted model that matches the drift type of the input image. We make three observations from the results. First, before adaptation, the detector is rather accurate in catching most drift sources but exhibits some noise. Second, when it uses the appropriately-adapted model, it is less likely to detect the data as drift and exhibits the same detection probability for drifted data and clean data. Therefore, Nazar is not expected to frequently detect root causes after it has adapted to them. Third, when the adaptation is less successful due to the drift, i.e., severity levels in this experiment, between the data which the model is adapted on and the new arriving data, the detection rates will remain high. This helps Nazar to continuously detect root causes that it failed to adapt to.



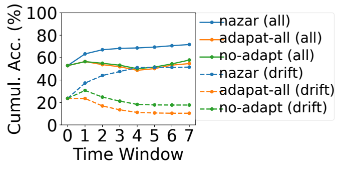
(a) Average accuracy.



(b) Average accuracy of drifted data.

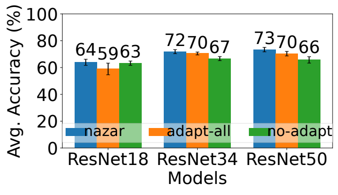


(c) No. of model versions over time.

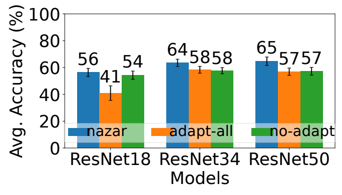


(d) ResNet18 accuracy over time.

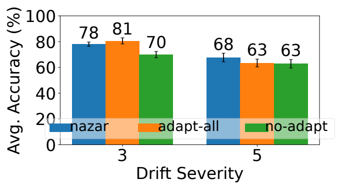
Figure 8. Experiments on cityscapes dataset. 8a and 8b show the average accuracy with three strategies. 8d shows the trace of accumulated accuracy per time window. 8c shows the number of model versions on user devices without set reduction and counterfactual analysis. Accuracy numbers in the figures are rounded to integers due to space constraint.



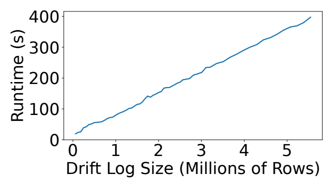
(a) Average accuracy (S = 3).



(b) Average accuracy (S = 5).



(c) Average accuracy (α = 1).



(d) Root cause analysis scalability.

Figure 9. Results of experiments on Animals dataset. 9a and 9b show the average accuracy with different drift severities. 9c shows the average accuracy with class skew of α=1. 9d shows the scalability of Nazar's root analysis. Accuracy numbers in the figures are rounded to integers due to space constraint.

5.7 End-to-End Workloads (Q5)

We now evaluate Nazar end-to-end on our streaming workloads. We first evaluate the general performance on the cityscapes dataset and then test them on the more synthetic animals dataset under more severe drift conditions, i.e., higher severity for weather drifts and class skew.

Average accuracy. We measure the average accuracy on all data averaged across the last 7 time windows on the cityscapes dataset in Figure 8a. For all model architectures, Nazar yields the highest average accuracies and the smallest standard deviations. The improvements in average accuracy compared with adapt-all are very significant: 10.1–19.4%.

We also measure the average accuracy of only the drifted data averaged across three types of data drift in Figure 8b, which shows even more significant accuracy improvements compared to adapt-all (49.5% on ResNet18 and 37.6% on ResNet34), which is because smaller models have a weaker ability to generalize on a mixed source of distributions.

Figure 8d shows the cumulative average accuracy over time for all data and for drifted ones. The cumulative accuracy of Nazar continuously improves over time with each adaptation, while the one for adapt-all increases for one adaptation window, as the drift level is mild even in the no-adapt case, and then decreases in the following 3 adaptation windows, after which they slightly increase. This demonstrates the utility of adapt by-cause, which provides much more stable accuracy over time, both for clean and drifted data.

Benefit of set reduction and counterfactual analysis. To find out if the set reduction and counterfactual analysis are effective in the end-to-end setting, we only run the FIM algorithm for root cause analysis, and find the average accuracy drops 1.3–9.7%. Meanwhile, the number of stored BN versions (recall we only adapt the BN layers, so each BN version represents a different adapted model) is much higher than Nazar, due to the redundant root causes. We show the numbers of BN versions with ResNet18 that are stored on user devices during each time window for FIM and Nazar in Figure 8c. The number of BN versions with Nazar is steady at 3 from the second time window. Note in this experiment we do not cap the number of model versions for Nazar.

Adaptation frequency. We evaluate whether the number of adaptation cycles affects overall performance. We split the end-to-end workloads into 4 intervals instead of 8 and found the results stay consistent. The average accuracy across the three models improved by 1.2-3.8%.

Drift severity. We vary the severity of the corruptions on the animal dataset (Figure 9a and 9b), where a higher severity represents a higher level of drift, and severity is denoted by "S". When severity is higher, all three methods degrade on both accuracy metrics but Nazar still outperforms among them. Nazar's improvements compared to adapt-all is larger when the severity is higher (3.8–10.4% better).

Class skew. We evaluate the effect of class skew on Nazar with ResNet50, by setting the Zipfian parameter α = 1. The results are presented in Figure 9c. Nazar fails to outperform adapt-all on average accuracy when severity is set to 3 on the animal dataset, because Nazar doesn't recognize class skew and thus can't adjust models for species with lower accuracies.Nazar also has a narrower view of diverse image features than adapt-all, when the variability of classes in each location is largely constrained by class skew, and thus tends to overfit the model during adaptation. When we increase the variety of images during each adaptation by splitting the workload into 4 intervals instead of 8, Nazar outperforms both baselines and improves the average accuracy of 0.9% compared with adapt-all. In addition, when the severity is higher, Nazar yields better average accuracy even when the number of time windows is 8, also shown in Figure 9c, which offsets the impact from class skew.

5.8 Runtime and Scalability (Q6)

Runtime. Once a sufficient number of entries contain data drift, Nazar should be able to generate adaptations in a reasonably timely manner to respond to the drift. We measure the latency of Nazar from the invocation of the root cause analysis function, through the model adaptations and up until the adapted models are written in S3, and repeat the experiment four times. The average end-to-end latency is 50 minutes. The average root cause analysis runtime is only 46 seconds, while the rest of the time is consumed by the model adaptation. Note that this time can be shortened by adding additional or larger adaptation instances.

Scalability. Nazar's three components are designed to be highly scalable. The drift log relies on a scalable database (Aurora), and model adaptation can be easily parallelized on a large number of instances. The only component that could potentially become a scalability bottleneck is the root cause analysis function, since a single function needs to operate on a relatively large number of Aurora rows via a series of SQL queries. Figure 9d presents the runtime of the root cause analysis running on a single instance, as a function of the drift log size. We can see the relationship between the runtime and the number of rows in the drift log is completely linear. The reason is that the FIM algorithm is linear [4] and the number of candidate sets of attributes is greatly reduced for counterfactual analysis after the set reduction.

6 Conclusions

Nazar is the first end-to-end monitoring and adaptation system for mobile ML deployments. Nazar chains on-device drift detection with root cause analysis and by-cause adaptation run in the cloud, which provides high accuracy, without user input. Interesting avenues for future work are adapting Nazar to distributed federated learning, and developing techniques for improved user privacy.

Acknowledgments

We thank the anonymous reviewers for their insightful feedback which substantially improved our paper. Our work was supported in part by awards from Google, Amazon, and Meta; and by NSF awards CNS-2106530, CNS-2143868.

References

[1] Firas Abuzaid, Peter Kraft, Sahaana Suri, Edward Gan, Eric Xu, Atul Shenoy, Asvin Ananthanarayan, John Sheu, Erik Meijer, Xi Wu, Jeff Naughton, Peter Bailis, and Matei Zaharia. 2018. DIFF: A Relational Interface for Large-Scale Data Explanation. *Proc. VLDB Endow.* 12, 4 (dec 2018), 419–432. https://doi.org/10.14778/3297753.3297761

[2] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data* (Washington, D.C., USA) *(SIGMOD '93)*. Association for Computing Machinery, New York, NY, USA, 207–216. https://doi.org/10.1145/170035.170072

[3] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, A Inkeri Verkamo, et al. 1996. Fast discovery of association rules. *Advances in knowledge discovery and data mining* 12, 1 (1996), 307–328.

[4] Rakesh Agrawal, Ramakrishnan Srikant, et al. 1994. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, Vol. 1215. Santiago, Chile, 487–499.

[5] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. 2022. Ekya: Continuous Learning of Video Analytics Models on Edge Compute Servers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 119–135. https://www.usenix.org/conference/nsdi22/presentation/bhardwaj

[6] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. 2016. End to End Learning for Self-Driving Cars. *CoRR* abs/1604.07316 (2016). arXiv:1604.07316 http://arxiv.org/abs/1604.07316

[7] Christian Borgelt. 2003. Efficient implementations of apriori and eclat. In *FIMI'03: Proceedings of the IEEE ICDM workshop on frequent itemset mining implementations*. Citeseer, 90.

[8] Christian Borgelt. 2005. An Implementation of the FP-growth Algorithm. In *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*. 1–5.

[9] Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. 2019. Federated learning of out-of-vocabulary words. *arXiv preprint arXiv:1903.10635* (2019).

[10] Eyal Cidon, Evgenya Pergament, Zain Asgar, Asaf Cidon, and Sachin Katti. 2021. Characterizing and Taming Model Instability Across Edge Devices. *Proceedings of Machine Learning and Systems* 3 (2021).

[11] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. 2016. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[12] Edward B Fowlkes and Colin L Mallows. 1983. A method for comparing two hierarchical clusterings. *Journal of the American statistical association* 78, 383 (1983), 553–569.

[13] Robert M French. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences* 3, 4 (1999), 128–135.

[14] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. 2020. An efficient framework for clustered federated learning. *Advances in Neural Information Processing Systems* 33 (2020), 19586–19597.

[15] Peizhen Guo, Bo Hu, and Wenjun Hu. 2021. Mistify: Automating DNN Model Porting for On-Device Inference at the Edge.. In *NSDI*. 705–719.

[16] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. *ACM sigmod record* 29, 2 (2000), 1–12.

[17] Wei Hao, Aahil Awatramani, Jiayang Hu, Chengzhi Mao, Pin-Chun Chen, Eyal Cidon, Asaf Cidon, and Junfeng Yang. 2022. A Tale of Two

Models: Constructing Evasive Attacks on Edge Models. *Proceedings of Machine Learning and Systems* 4 (2022), 414–429.

[18] Wei Hao, Daniel Mendoza, Rafael Mendes, Deepak Narayanan, Amar Phanishayee, Asaf Cidon, and Junfeng Yang. 2024. MGit: A Model Versioning and Management System. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*, Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.). PMLR, 17597–17615. https://proceedings.mlr.press/v235/hao24c.html

[19] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. 2020. FedML: A Research Library and Benchmark for Federated Machine Learning. *Advances in Neural Information Processing Systems* (2020).

[20] Dan Hendrycks and Thomas Dietterich. 2019. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. *Proceedings of the International Conference on Learning Representations* (2019).

[21] Dan Hendrycks and Kevin Gimpel. 2017. A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. In *International Conference on Learning Representations*.

[22] Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. 2019. Deep Anomaly Detection with Outlier Exposure. *Proceedings of the International Conference on Learning Representations* (2019).

[23] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. 2019. Using self-supervised learning can improve model robustness and uncertainty. *Advances in neural information processing systems* 32 (2019).

[24] Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie. 2018. The iNaturalist Species Classification and Detection Dataset. In *CVPR*. https://arxiv.org/abs/1707.06642

[25] Yen-Chang Hsu, Yilin Shen, Hongxia Jin, and Zsolt Kira. 2020. Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10951–10960.

[26] iNaturalist. 2023. iNaturalist. https://www.inaturalist.org/.

[27] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. pmlr, 448–456.

[28] kaggle,2020 2020. Historical Daily Weather Data 2020. https://www.kaggle.com/datasets/vishalvjoseph/weather-dataset-for-covid19-predictions. Accessed: Apr 12, 2021.

[29] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. 2022. Fedscale: Benchmarking model and system performance of federated learning at scale. In *International Conference on Machine Learning*. PMLR, 11814–11827.

[30] Fan Lai, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. 2021. Oort: Efficient Federated Learning via Guided Participant Selection. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, 19–35. https://www.usenix.org/conference/osdi21/presentation/lai

[31] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. 2018. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in neural information processing systems* 31 (2018).

[32] David Lewis. 2013. *Counterfactuals*. John Wiley & Sons.

[33] Siyuan Li, Iago Breno Araujo, Wenqi Ren, Zhangyang Wang, Eric K. Tokuda, Roberto Hirata Junior, Roberto Cesar-Junior, Jiawan Zhang, Xiaojie Guo, and Xiaochun Cao. 2019. Single Image Deraining: A Comprehensive Benchmark Analysis. *IEEE Conference on Computer Vision and Pattern Recognition* (2019).

[34] Zhizhong Li and Derek Hoiem. 2017. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence* 40, 12 (2017), 2935–2947.

[35] Shiyu Liang, Yixuan Li, and R. Srikant. 2018. Enhancing The Reliability of Out-of-distribution Image Detection in Neural Networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=H1VGkIxRZ

[36] Edo Liberty, Zohar Karnin, Bing Xiang, Laurence Rouesnel, Baris Coskun, Ramesh Nallapati, Julio Delgado, Amir Sadoughi, Yury Astashonok, Piali Das, et al. 2020. Elastic machine learning algorithms in amazon sagemaker. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 731–737.

[37] Weitang Liu, Xiaoyun Wang, John Owens, and Yixuan Li. 2020. Energy-based out-of-distribution detection. *Advances in neural information processing systems* 33 (2020), 21464–21475.

[38] Davide Maltoni and Vincenzo Lomonaco. 2019. Continuous learning in single-incremental-task scenarios. *Neural Networks* 116 (2019), 56–73.

[39] Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. Vol. 24. Elsevier, 109–165.

[40] Akshay Naresh Modi, Chiu Yuen Koo, Chuan Yu Foo, Clemens Mewald, Denis M. Baylor, Eric Breck, Heng-Tze Cheng, Jarek Wilkiewicz, Levent Koc, Lukasz Lew, Martin A. Zinkevich, Martin Wicke, Mustafa Ispir, Neoklis Polyzotis, Noah Fiedel, Salem Elie Haykal, Steven Whang, Sudip Roy, Sukriti Ramesh, Vihan Jain, Xin Zhang, and Zakaria Haque. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *KDD 2017*.

[41] NannyML 2023. NannyML (release 0.9.1). https://github.com/NannyML/nannyml. NannyML, Belgium, OHL..

[42] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural networks* 113 (2019), 54–71.

[43] Matthias Paulik, Matt Seigel, Henry Mason, Dominic Telaar, Joris Kluivers, Rogier van Dalen, Chi Wai Lau, Luke Carlson, Filip Granqvist, Chris Vandevelde, et al. 2021. Federated evaluation and tuning for on-device personalization: System design & applications. *arXiv preprint arXiv:2102.08503* (2021).

[44] Jian Pei, Jiawei Han, Runying Mao, et al. 2000. CLOSET: An efficient algorithm for mining frequent closed itemsets.. In *ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, Vol. 4. 21–30.

[45] Jian Pei, Jiawei Han, B. Mortazavi-Asl, H. Pinto, Qiming Chen, U. Dayal, and Mei-Chun Hsu. 2001. PrefixSpan,: mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings 17th International Conference on Data Engineering*. 215–224. https://doi.org/10.1109/ICDE.2001.914830

[46] PyTorch. 2023. PyTorch Android Examples. https://github.com/pytorch/android-demo-app. Accessed: May 3, 2023.

[47] Hang Qiu, Ioanna Vavelidou, Jian Li, Evgenya Pergament, Pete Warden, Sandeep Chinchali, Zain Asgar, and Sachin Katti. 2022. ML-EXray: Visibility into ML deployment on the edge. *Proceedings of Machine Learning and Systems* 4 (2022), 337–351.

[48] Stephan Rabanser, Stephan Günnemann, and Zachary Lipton. 2019. Failing loudly: An empirical study of methods for detecting dataset shift. *Advances in Neural Information Processing Systems* 32 (2019).

[49] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. 2019. Federated learning for emoji prediction in a mobile keyboard. *arXiv preprint arXiv:1906.04329* (2019).

[50] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. 2017. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2001–2010.

[51] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla,

Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[52] David Sculley. 2010. Web-scale K-means clustering. In *Proceedings of the 19th international conference on World wide web*. 1177–1178.

[53] Shreya Shankar, Labib Fawaz, Karl Gyllstrom, and Aditya G Parameswaran. 2023. Moving Fast With Broken Data. *arXiv preprint arXiv:2303.06094* (2023).

[54] Shreya Shankar, Rolando Garcia, Joseph M. Hellerstein, and Aditya G. Parameswaran. 2024. Operationalizing Machine Learning: An Interview Study. In *The 27th ACM Conference on Computer-Supported Cooperative Work and Social Computing*.

[55] Shreya Shankar, Bernease Herman, and Aditya Parameswaran. 2022. Rethinking Streaming Machine Learning Evaluation. *ICLR* (2022).

[56] Shreya Shankar and Aditya G. Parameswaran. 2022. Towards Observability for Production Machine Learning Pipelines. *Proc. VLDB Endow.* 15, 13 (sep 2022), 4015–4022. https://doi.org/10.14778/3565838.3565853

[57] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. 2017. Incremental learning of object detectors without catastrophic forgetting. In *Proceedings of the IEEE international conference on computer vision*. 3400–3409.

[58] Jihoon Tack, Sangwoo Mo, Jongheon Jeong, and Jinwoo Shin. 2020. Csi: Novelty detection via contrastive learning on distributionally shifted instances. *Advances in neural information processing systems* 33 (2020), 11839–11852.

[59] TensorFlow. 2023. TensorFlow Lite. https://www.tensorflow.org/lite.

[60] Weather Underground. 2023. Weather Underground. https://www.wunderground.com/. Accessed: Mar 8, 2023.

[61] Arnaud Van Looveren, Janis Klaise, Giovanni Vacanti, Oliver Cobb, Ashley Scillitoe, and Robert Samoilescu. 2019. *Alibi Detect: Algorithms for outlier, adversarial and drift detection*. https://github.com/SeldonIO/alibi-detect

[62] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. 2017. Amazon Aurora: Design considerations for high throughput cloud-native relational databases. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1041–1052.

[63] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. 2021. Tent: Fully Test-Time Adaptation by Entropy Minimization. In *International Conference on Learning Representations*. https://openreview.net/forum?id=uXl3bZLkr3c

[64] Ewen Wang, Boyi Chen, Mosharaf Chowdhury, Ajay Kannan, and Franco Liang. 2023. Flint: A platform for federated learning integration. *Proceedings of Machine Learning and Systems* 5 (2023), 21–34.

[65] Gerhard Widmer and Miroslav Kubat. 1996. Learning in the presence of concept drift and hidden contexts. *Machine learning* 23 (1996), 69–101.

[66] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, et al. 2019. Machine learning at Facebook: Understanding inference at the edge. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 331–344.

[67] Xi Yin, Xiang Yu, Kihyuk Sohn, Xiaoming Liu, and Manmohan Chandraker. 2019. Feature Transfer Learning for Face Recognition With Under-Represented Data. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), 5697–5706. https://api.semanticscholar.org/CorpusID:102490877

[68] Marvin Zhang, Sergey Levine, and Chelsea Finn. 2022. Memo: Test time robustness via adaptation and augmentation. *Advances in neural information processing systems* 35 (2022), 38629–38642.