LEAF: Navigating Concept Drift in Cellular Networks

Shinan Liu, Francesco Bronzino, Paul Schmitt, Arjun Nitin Bhagoji, Nick Feamster, Hector Garcia Crespo, Timothy Coyle, Brian Ward

LEAF: Navigating Concept Drift in Cellular Networks

SHINAN LIU, University of Chicago, USA
FRANCESCO BRONZINO, Univ Lyon, EnsL, UCBL, CNRS, LIP, France
PAUL SCHMITT, University of Hawaii, Manoa, USA
ARJUN NITIN BHAGOJI, University of Chicago, USA
NICK FEAMSTER, University of Chicago, USA
HECTOR GARCIA CRESPO, Verizon, USA
TIMOTHY COYLE, Verizon, USA
BRIAN WARD, Verizon, USA

Operational networks commonly rely on machine learning models for many tasks, including detecting anomalies, inferring application performance, and forecasting demand. Yet, model accuracy can degrade due to *concept drift*, where either the relationships between features and the target to be predicted, or the features themselves change. Mitigating concept drift is an essential part of operationalizing machine learning models in general, but is of particular importance in networking's highly dynamic deployment environments. In this paper, we first characterize concept drift in a large cellular network for a major metropolitan area in the United States. We find that concept drift occurs across many important key performance indicators (KPIs), independently of the model, training set size, and time interval—thus necessitating practical approaches to detect, explain, and mitigate it. We then show that frequent model retraining with newly available data is not sufficient to mitigate concept drift, and can even degrade model accuracy further. Finally, we develop a new methodology for concept drift mitigation, Local Error Approximation of Features (LEAF). LEAF works by detecting drift; explaining the features and time intervals that contribute the most to drift; and mitigating it using forgetting and over-sampling. We evaluate LEAF against industry-standard mitigation approaches (notably, periodic retraining) with more than four years of cellular KPI data. Our tests with a major cellular provider in the US show that LEAF consistently outperforms periodic and triggered retraining on complex, real-world data while reducing costly retraining operations.

CCS Concepts: • **Networks** → **Network measurement**; **Error detection and error correction**; **Mobile networks**; • **Computing methodologies** → **Learning under covariate shift**.

Additional Key Words and Phrases: concept drift, cellular network, regression, explanation, mitigation

1 INTRODUCTION

Network operators rely on machine learning (ML) models to perform many tasks, including anomaly detection [52, 53], performance inference [21] and diagnosis, and forecasting [13, 33, 44]. Unfortunately, deploying and maintaining these models can prove challenging in practice [55]. One significant operational challenge is *concept drift*, whereby changes to the data distribution (*virtual drift*) or its relationship with the target to be predicted (*real concept drift*), deteriorate model performance over time [22, 36]. Previous work in applying ML models to network management tasks has typically trained and evaluated models on fixed, offline datasets [6, 13, 16, 44, 52–54], demonstrating the ability to predict various network features at fixed points in time on a static dataset. Yet, a model that performs well offline on a single dataset may not in fact perform well in practice, especially over time as characteristics change.

Authors' addresses: Shinan Liu, University of Chicago, Chicago, IL, USA, shinanliu@uchicago.edu; Francesco Bronzino, Univ Lyon, EnsL, UCBL, CNRS, LIP, Lyon, France, francesco.bronzino@ens-lyon.fr; Paul Schmitt, University of Hawaii, Manoa, Honolulu, HI, USA, pschmitt@hawaii.edu; Arjun Nitin Bhagoji, University of Chicago, Chicago, IL, USA, abhagoji@uchicago.edu; Nick Feamster, University of Chicago, Chicago, IL, USA, feamster@uchicago.edu; Hector Garcia Crespo, Verizon, North Richland Hills, TX, USA, hector.garcia@verizon.com; Timothy Coyle, Verizon, Chicopee, MA, USA, timothy.coyle@verizon.com; Brian Ward, Verizon, Fort Worth, TX, USA, brian.ward2@verizon.com.

Concept drift is a relatively well-understood phenomenon in ML for other prediction problems (*e.g.*, image and text classification [22, 36, 50]). Yet, mitigating concept drift for networking problems, such as forecasting Key Performance Indicators (KPIs) in a cellular network introduces fundamentally new challenges that make previous approaches from other domains inapplicable. Networks have unique characteristics, such as dynamic signal interference due to environment changes (*e.g.*, weather, seasonality, etc.) [64], which calls for new approaches. In contrast to previous tasks, where the semantics of prediction occurs on a fixed object and characteristics of the features change relatively slowly over time [18, 22, 65], predictions of network characteristics occur continuously and occur within the context of a system that changes over time due to periodicity (*e.g.*, seven-day period of volume), gradual evolution (*e.g.*, the constant addition of capacity by new equipment installations), and exogenous shocks (*e.g.*, a software upgrade or a sudden change in traffic patterns such as the COVID-19 pandemic which resulted in significant changes in user behaviors [34, 37]).

Beyond simply detecting concept drift, operators may also want to interpret why a model has become less accurate and mitigate it. Previous research has developed explanation methods for concept drift in classification problems [27, 61], but prediction in cellular networks is often a regression problem. The absence of distinct classes rules out the direct use of existing methods. In addition, the subtleties of drift in regression make its impact more nuanced. To this end, we develop *LEAF* (Local Error Approximation of Features), which (1) uses explainability techniques to identify features that contribute the most to drift; and (2) mitigates drift through triggered, focused re-retraining, forgetting, and over-sampling approaches that leverage these identified features. This paper makes three contributions:

First, **we characterize concept drift in the context of a large cellular network**, exploring drift for KPIs using more than four years of KPI data in a major United States city and surrounding metropolitan area, from one of the largest cellular providers in the United States. We demonstrate concept drift in a large cellular network comparing different KPIs, model families, training set sizes, and periods across many regression models and tasks. Concept drift occurs consistently and independently of both the size and period of the training set. The diverse, longitudinal nature of the dataset used presents a challenging concept drift problem.

To detect, mitigate, and explain concept drift in cellular networks, **we introduce *LEAF* (Local Error Approximation of Features)** as an explainable ML approach for networking models. We apply Kolmogorov-Smirnov Windowing (KSWIN) to time-series of estimated errors, which tells us when a model drifts. We find the most representative features that reflect error distributions, and use LEAplot and LEAgram to inform operators about for *which features*, *where*, and *how much* concept drift occurs, which maps to the under-trained region of a model. Based on these explanations, the framework strategically re-samples the training data, and creates temporal ensembles to mitigate drift. In addition to providing explanations for drift, LEAF also outperforms the baseline approach of regular retraining using a fixed-size window of recent observations, which we find can actually harm model performance in certain settings.

Finally, **we evaluate the effectiveness and efficiency of LEAF holistically on the complete dataset and showcase the explanation power**, across boosting, bagging, and LSTM-based models and a variety of forecasting targets. LEAF consistently outperforms existing mitigation approaches, while reducing costly retraining operations by as much as 76.9% (compared to periodic retraining). Although KPIs with higher dispersion are harder to mitigate, LEAF can still consistently reduce errors. From a case study, LEAF provides possible explanations of drift by identifying the features most responsible for drift. LEAplot further localizes the high-error eNodeBs, where the top 5% of error mostly comes from suburban areas. From LEAgram, we also find that over- and under-estimation changes over time, and overestimation exhibits high correlation with less mobility.

Table 1. Summary of the datasets and characteristics of target KPIs. DVol: Downlink volume; PU: Peak active UEs; DTP: Downlink Throughput; REst: RRC establishment success; CDR: S1-U call drop rate; GDR: RTP gap duration ratio.

Collection period	Jan. 1st 2018 – March 28th 2022					
Identifiers	eNodeB ID & Time stamp					
Number of eNBs	Fixed Dataset: 412 common eNBs Evolving Dataset: 898 eNBs					
Number of logs	Fixed Dataset: 699,381 Evolving Dataset: 1,084,837					
Number of KPIs	224					
Groups of KPIs (Target KPIs within group)	Resource utilization (DVol, PU) Network performance (DTP, REst) User experience (CDR, GDR)					
KPI	DVol	PU	DTP	REst	CDR	GDR
Std/Mean	0.81	1.76	0.59	0.85	2.48	8.52

To the best of our knowledge, LEAF is the first method to provide explanations for concept drift for regression, building on previous work that has largely focused on classification problems [27, 61]. We thus envision LEAF being applied beyond cellular networks, to other network management problems that use black-box models for regression-based prediction. Our belief in the generalizability of LEAF arises from the diversity in time series across KPIs and time that LEAF tackles successfully. Such avenues present rich opportunities for future work. Towards this goal and for reproducibility purposes, we release a subset of our dataset to encourage future work from the community [1].

2 PROBLEM SETUP

The main problem we tackle in this paper is that of accurate longitudinal prediction in the presence of concept drift. This is a challenging problem that requires a dataset with sufficient length and variation to determine if our proposed methods would be effective in a real-world setting. In this section, we describe the diverse, longitudinal dataset used in this paper. We then define our time-series regression problem, which is to predict the current value of a target KPI variable given historical data. Finally, we detail the metrics we use throughout the paper to measure performance.

2.1 Dataset Description

Our analysis in this paper is based on more than four years (January 1, 2018 to March 28th, 2022) of daily measurements of LTE cellular network performance indicators collected at the eNodeB-level (evolved NodeBs, or the "base station" in 4G LTE) from a major wireless carrier in the United States. Table 1 summarizes the characteristics of the datasets, which we refer to as Fixed Dataset (the dataset that contains a fixed number of eNodeBs each day) and Evolving Dataset (an expanded set of the Fixed Dataset with a growing number of eNodeBs) in the remainder of the paper. [1]

Why these datasets? The first requirement for any dataset used to analyze concept drift is that it should actually contain clear evidence of drift. Our analysis in Section 3 establishes that the variables, which are Key Performance Indicators (KPIs) from the cellular network, in both the Fixed Dataset and the Evolving Dataset do indeed drift over time. Intuitively, this drift occurs

[1]We release a normalized version of the Fixed Dataset spanning from from May 2019 to May 2020 [1].

due to endogenous reasons like changes in network infrastructure and KPI definitions, as well as exogenous ones such as the COVID-19 pandemic.

The Fixed Dataset is affected by factors from software upgrades to mobility pattern changes. It provides an "apples to apples" comparison across time. It is diverse since different KPIs exhibit drastically different behavior over time. This diversity and the real-world nature of the dataset lead to generalizable insights from our methods, since a wide variety of possible time-series behavior is captured within our dataset. The Evolving Dataset extends the KPI diversity further by including the operational growth of eNodeBs in this area, increasing the level of heterogeneity in the dataset.

Further details. The Fixed Dataset contains information from 412 common eNodeBs across time. It is collected in a large city and surrounding metropolitan area (rural, suburban, and urban included) in the United States. The dataset spans more than four years—from January 1, 2018 to March 28, 2022—and contains 699,381 daily eNodeB-level logs. The Evolving Dataset includes the same information, but with a maximum of 898 eNodeBs, containing 1,084,837 daily logs. It covers all the eNodeBs within the tested region.

Each log contains 224 Key Performance Indicators (KPIs) collected for a base station on a particular date. KPIs are statistics collected and used by the operator of the network to monitor and assess network performance. The 224 KPIs fall into three categories: (1) resource utilization (*e.g.*, data volume, peak active users, active session time, cell availability rate), (2) access network performance (*e.g.*, throughput, connection establishment success, congestion, packet loss), and (3) user experience features (*e.g.*, call drop rate, RTP gap duration ratio, abnormal UE releases). Further, some of the KPIs have separate directional measurements.

2.2 Modeling Goal

Forecasting Problem. We focus our study of concept drift in the context of network forecasting. Network forecasting (load, performance, user experience) is an important problem for operators as it sets the foundation to guide infrastructure configuration [29], management [51], and augmentation [49, 59]. We focus on per-eNodeB level KPI forecasting which can be used as a foundation for capacity adjustment, deployment, maintenance, and operation in large cellular networks (not a focus of this paper). The nature of this problem is regression with time-series information. Regression models are a better fit than classification because we aim to provide fine-grained forecasting of numerical KPIs that can have wide ranges.

In line with multivariate regression modeling used in previous network forecasting applications [29, 49, 51, 59], which uses time-series of KPI histories, we use historical data—*i.e.*, all available KPIs and dates (as features) up to a given day—to forecast one or more target KPIs of interest 180 days in the future. We employ a 180-day forecast window, as operators need this duration for planning and executing long-term network infrastructure augmentation. This 180-day gap also makes it more challenging to explain and mitigate drift.

Forecasting Targets. We select as forecasting targets six KPIs out of the available 224 based on their measurement goals. We focus two KPIs out of each of the three groups that are most relevant to network planning: measurements of resource utilization (downlink data volume, peak number of active UEs / User Equipment), network performance (downlink throughput, RRC / Radio Resource Control establishment success), and user experience (S1-U / S1 User plane external interface call drop rate, RTP gap duration ratio). They not only cover the RAN events and wireless connections, but also the UE behaviors.

Additionally, the different focus of KPIs also makes them have different statistical behaviors. Table 1 summarizes the coefficients of variation (or dispersion) of these KPIs. These KPIs exhibit a variety of statistical patterns and characteristics which, as we will see in Section 3 and 6 can

ultimately affect how models drift over time, as well as the best strategies for mitigating drift for a particular KPI. All KPIs exhibit 7-day periodicity, although some KPIs exhibit far more variance than others. *Downlink volume (DVol)* and *RRC establishment success (REst)* present similar ranges in their distributions. In contrast, *S1-U call drop rate (CDR)* and *peak active UEs (PU)* show a more bursty behavior over time. While the data distributions of *downlink throughput (DTP)* and PU have balanced distributions that do not present a long tail, DVol, REst, and CDR present more skewed distributions.

2.3 Metrics for Evaluating Drift

The characterization of model performance over time is essential to evaluate concept drift. Given that we model network forecasting as a regression problem, we test model effectiveness by computing distances between prediction and ground truth by date, specifically using Root Mean Squared Error (RMSE). To better understand drift across different KPIs whose natural operating value ranges are drastically different (*e.g.*, call drop rates are scalars mostly less than 1, while downlink volume scalars are often greater than 300,000), we normalize the RMSE by maxmin and derive the Normalized Root Mean Squared Error (NRMSE).

In addition to offering an equitable comparison across multiple KPIs, the NRMSE is well-suited to understand concept drift because (1) it penalizes large errors that can be costly for ISPs' operations (*e.g.*, load forecasting errors can lead to over-provision); and (2) it captures the relative impact of errors over extended periods of time. In practice, NRMSE scores under 0.1 and R^2 over 90% indicate that the regression model has very good prediction power [45]. [2]

Average NRMSE distance from a static model. To show the long-term effectiveness of different mitigation schemes, we study the relative evolution of errors over extended periods of time. For each day in the dataset, we compute the average NRMSE across all eNodeBs and compare it to the error generated by a model that is never retrained, *i.e.*, a static model. We define $\Delta\overline{NRMSE}$ as the average distance between the error for a mitigated model M_1 against the static model M_0 in the form of a percentage distance. We define:

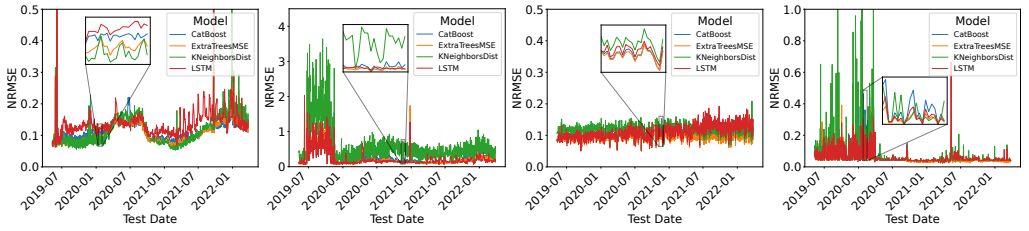
$$\Delta\overline{NRMSE}(M_1, M_0) = \frac{\overline{NRMSE}(M_1) - \overline{NRMSE}(M_0)}{\overline{NRMSE}(M_0)} \times 100\% \quad (1)$$

where $\overline{NRMSE}$ is the average over time.

3 DRIFT CHARACTERIZATION

In this section, we explore how trained forecasting models are affected by concept drift. We train a number of models from different categories of regression-based predictors, targeting different KPIs, and different training parameters. After identifying concept drift behavior in the cellular network dataset we use, we evaluate the effectiveness of periodic retraining with recent data, which is the state of the art for drift mitigation. We demonstrate that it is hard to identify a single retraining strategy across models and KPIs, making naïve retraining a difficult strategy to implement in practice. All measurements in this section use the Evolving Dataset.

[2]To avoid possible errors in interpretation, we evaluate our results using other metrics including coefficient of determination, mean absolute percentage error, mean absolute error, explained variance score, Pearson correlation, mean squared error, and median absolute error. We omit these metrics for brevity. We observe that all phenomena we describe in the paper using NRMSE also hold for these metrics.



(a) Volume. (b) Peak active UEs. (c) Throughput. (d) Gap duration ratio.

Fig. 1. Drift of different models for KPIs of interest. Inset figures exhibit a 3-week view (all starting from Sunday) of NRMSE for the box-selected period. Some data is lost between July, 2019 and January, 2020 for Peak active UEs. Note that the y-axes are scaled to different range to accommodate larger errors in Fig.1b, 1d.

3.1 Model Selection

To explore the performance of different widely adopted regression techniques, we use for all experiments in this paper the AutoGluon [2] pipeline and TensorFlow. AutoGluon is used to quickly prototype deep learning and machine learning algorithms on various existing frameworks. We emphasize that the goal of our study is not to create and tune models that maximize performance, but rather to demonstrate and better understand concept drift in a real-world network.

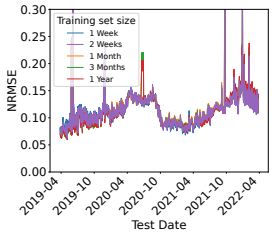
We select *four different families of models*: (1) gradient boosting algorithms like LightGBM, LightGBMLarge, LightGBMXT, CatBoost, and XGBoost; (2) bagging algorithms such as Random Forest and Extra Trees; (3) distance-based algorithms like KNeighbors; (4) recurrent neural networks such as LSTM. All models either incorporate temporal features (*e.g.*, time stamps, day of the week, month, year), or are time-series models (LSTM). Although it is feasible to fine-tune each model's hyperparameters by hand, we rely on the auto-selection pipeline with the goal of a fair comparison and to make training scalable and efficient. Take for instance, our LSTM network, built using TensorFlow. It has 100 LSTM units, a 0.2 dropout rate, and a dense output layer, processing inputs reshaped to 'samples, 7-day timesteps, KPI features'.

For all experiments, we develop models for each selected target KPI. As the input of the models, we use a portion of the history of all categorical and numerical KPIs from all eNodeBs up to the date when the model is generated. We generate a single model for the entire network, i.e., we create a model capable of forecasting values for samples collected from each base station.

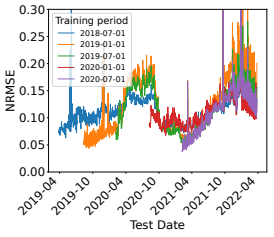
3.2 Comparing Drift Behavior across KPIs

We explore whether we can observe drift for the KPI forecasting task and additionally whether the drift patterns from different KPIs are similar. We train all the models mentioned above for each target KPI, while keeping the inputs of models unchanged. We use a 90-day window of historical data from all eNodeBs with an end date of July 1, 2018 for our training data, *i.e.*, forecasting KPIs starting from December 28th, 2018 (we plot from Mid-March 2019 because of data losses between January and March 2019). We then test these models on data subsets split by date. Note that the number of samples evaluated each day may vary due to ongoing infrastructure expansions.

Uncorrelated KPIs exhibit different drift patterns. Figure 1 presents the concept drift across time for the three categories of KPIs (due to space limitations, we omit CDR and REst). Overall, the drift patterns are quite unique for each class, and they vary in two aspects. First, deviations in NRMSE occur at different periods of time. For instance, in Figure 1a, the NRMSE of downlink volume, specifically in the CatBoost model, experiences a substantial increase from 0.102 to 0.136



(a) Training set size.



(b) Training set period (train. set size = 2 weeks).

Fig. 2. Effects of training set size and training set period on concept drift of CatBoost models for Volume.

in April 2020 due to the COVID-19 lockdown—a clear example of sudden concept drift. However, it reverts back to more normal values, achieving an average of 0.089 by October 2020. From March 2021, the NRMSE starts to rise once more, peaking at 0.145 in January 2022. For the prediction of peak users, Figure 1b demonstrates that July 2019 to November 2019 are more challenging to predict (0.789 NRMSE for CatBoost), because of lost data. Moreover, short-lived, abrupt increases in error are more frequent than other KPIs, due to the burstiness of GDR. Second, the high-frequency components have different patterns for KPIs. Using signal processing techniques like STFT, we found no clear weekly pattern in the NRMSE of CDR and GDR, while other KPIs exhibit such patterns, as the 3-week insets show.

Different KPIs are most effectively predicted using different models. As shown in Figure 1, we can find a model that performs relatively well for each target KPI. For all of the target KPIs, there is at least one model with NRMSE less than 0.1 (indicating good prediction power [45]) for at least one year. For example, even for GDR, the KPI that is most challenging to accurately predict (due to high coefficient of variance / dispersion), the average NRMSE from ExtraTrees is 0.055. CatBoost for volume prediction has an average NRMSE of 0.116, with less than 0.1 on 473 days.

3.3 Analyzing Drift for a Single KPI

In this section, we analyze how changes in the model used, and how it is trained, impact concept drift. We use a single KPI (downlink volume) as an illustrative example.

Individual KPI predictions drift consistently across different models. As mentioned, we use four different well-trained models to check for concept drift over time, while predicting KPIs 180 days in the future. We find (Figure 1a) that all models exhibit the same pattern. For example, between April and October 2020, all models exhibit a drastic rise in NRMSE due to the COVID-19 pandemic. Such consistency is also found in the gradual increase in NRMSE after March 2021. We find similar model drift across other metrics such as R^2, mean absolute error, etc. This pattern holds for each individual KPI. We also perform a cross-model analysis of the most important features (as determined by permutation). In predicting downlink volume, all four models consistently considered historical downlink volumes and average UE downlink packets as significant. A similar pattern was observed for the other KPIs, with their top 5 crucial features largely converging (around 3-4 shared features), suggesting that these different models are effectively capturing similar feature relationships in this dataset. Given our observations that models perform similarly well, we use CatBoost for the rest of this paper to simplify the presentation.

Consistent drift appears when trained using different training set sizes. To investigate how parameters of a model could impact concept drift, we evaluate the impact of varying the training set size. We vary the size of historical data from eNodeBs with an end date of July 1st, 2018, and retrain models for each size training window. As Figure 2a illustrates, all NRMSE values of different training set window sizes drift similarly (within an NRMSE of 0.114 to 0.115 on average across training set size) over time. While training set windows of one week of data lead to a slightly higher NRMSE, and three months and one year witness a glitch around June 2020, the signal pattern remains the same: no matter what training set size, NRMSE experiences a sudden increase after COVID-19 lockdown and gradually recovers after October 2020, then increases again from March 2021 and peaks in January 2022.

In Figure 2a, we see that the model retraining interval of two weeks performs very similarly to that of one year, while the model training time on two weeks is 18x more efficient than one year. Given these results, we use two weeks for the training set window for the rest of the paper if not otherwise specified. This choice optimizes the tradeoff between performance over time (*i.e.*, robustness) and efficiency.

3.4 Naïve Retraining in Practice

In operational networks, a common approach to counteract potential concept drift is to retrain models regularly. Retraining using the latest data is often considered an effective way to deal with concept drift. Many existing solutions [28, 56] adopt this approach, which outperforms recent dedicated drift mitigation methods [42, 62]. To understand the effectiveness of this approach for the Evolving Dataset, we retrain a number of different models using different retraining frequencies. Note that we do not tune the hyperparameters at each retrain, because it is very costly for exploring various combinations of hyperparameters to determine a better configuration and thus impractical, due to the need for continuous manual adjustments from human experts. For this experiment, we use a training set of 14 days to forecast traffic volume 180 days in the future, using the CatBoost model. Given a retrain frequency N, a model is retrained using the latest 14-day data. It is evaluated using the NRMSE for the next N days and is then replaced every N days.

Somewhat counter to conventional practice, we find that simply retraining the model at regular intervals is insufficient for efficiently combating concept drift for a diverse, longitudinal dataset. Naïve retraining is either less effective, or is effective but inefficient, requiring frequent retraining. In Table 2, we show the average NRMSE changes compared to the static model. For lower variance KPIs, *i.e.*, DVol, REst, and DTP, the more frequently a model is retrained, the better results we obtain. Unfortunately, while a 7-day retraining period can mitigate a large portion of errors, it is very costly for the network operator. Further, we observe that this trend does not exist for bursty, high-variance KPIs such as CDR (47.79% of error increased every 7 days) and GDR (retrain every 365 days is better than every 90 days). In this case, frequent retraining that simply naïvely uses all of the most recent data can even adversely affect model performances. In Figure 2b, we train on different 14-day windows of historical data from all eNodeBs to verify the pattern. It suggests that, *models trained on more recent time periods do not necessarily result in better model performance*, because the suddenly changed distribution is more similar to previous samples. For example, the model trained from July 1, 2018, yielded an NRMSE of 0.133 during lockdown, while the model trained from January 1, 2019, despite being more recent, resulted in a higher NRMSE of 0.169, indicating worse performance.

Intuitively, a key reason that naïve retraining may not work is that it is triggered at regular intervals, even though drift occurrences are irregular. It does not take into account when, where, and why drift is occurring. Thus, at times retraining is not necessary, or it is planned before drift actually occurs. This strategy ignores the fact that models trained on more recent periods do not

Table 2. Changes of average NRMSE and number of retrains, over time, for different periodic retraining strategies on CatBoost.

Retr. Period	$\Delta\overline{NRMSE}$ of Target KPIs						#Retr.
	DVol	PU	DTP	REst	CDR	GDR	
Static	–	–	–	–	–	–	0
7 days	−40.34%	−55.36%	−27.21%	−48.00%	47.79%	−0.38%	169
30 days	−30.66%	−43.73%	−21.40%	−40.12%	−0.75%	2.75%	39
90 days	−16.83%	−16.12%	−19.07%	−27.33%	7.89%	42.24%	13
180 days	−12.22%	−0.34%	−14.85%	−18.82%	−4.20%	76.28%	6
365 days	−2.27%	−5.13%	−10.65%	−11.53%	−5.97%	6.07%	3

necessarily result in better performance (Section 3.3). Further, complete data replacement neglects finer-grained error information across samples, throwing away useful samples from the past.

Overall, we conclude that although retraining is essential, naïvely performing it at regular intervals is not sufficient for efficient and explainable drift mitigation. It works best at high retraining frequencies and requires specific tests across different KPIs to at best tuning its performance. Both are challenging when run at scale in operational networks. Also, naïve retraining neglects finer-grained temporal error information across samples and thus loses explainability. This motivates our development of the LEAF framework in the rest of this paper.

4 LOCAL ERROR APPROXIMATION OF FEATURES (LEAF)

In this section, we introduce *LEAF*, our framework for drift detection, explanation, and mitigation. We leverage explainable AI methods to provide us with insights about concept drift to mitigate it effectively. We first describe the core intuition behind the LEAF framework, followed by the technical details for each component.

4.1 LEAF Framework Overview

LEAF is designed to work with any supervised regression model and provide explanations for black boxes. Based on the explanations provided, targeted mitigation strategies are applied to compensate for concept drift. Existing solutions face two limitations: (1) previous concept drift explanation and mitigation techniques are limited to classification problems (*e.g.*, [27, 61]); (2) concept drift mitigation is often coarse-grained, only focused on the global performance metrics (*e.g.*, [7]).

LEAF overcomes these limitations by implementing a pipeline of three components: (1) a drift *detector*; (2) a set of tools to *explain* drift for features; and (3) a drift *mitigator*. LEAF works in a black-box manner, only requiting access to the previously used training set data, new data as it arrives, and the generated model.

Figure 3 shows the three steps in LEAF's pipeline: First, the detector ingests the outputs of the model in the form of NRMSE time-series to determine whether drift is occurring. The detector applies the well-known Kolmogorov-Smirnov Windowing (KSWIN)[3] method [47, 57] on the time-series to identify a change in the distribution of the output error, providing an indicator of whether drift is occurring. Drift detection is critical but we mainly focus on drift explanation and informed mitigation in this paper, as there is significant prior work on detection. Further details on drift detection are in Appendix A.

[3]We apply the most effective drift detection techniques in this well-explored area [8, 9, 24, 36]. We also tested ADWIN, DDM, HDDM, EDDM, PageHinkley, but KSWIN was the most effective on our NRMSE series.

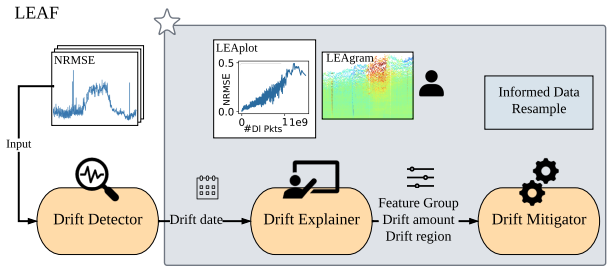


Fig. 3. The LEAF framework that detects (Appendix A), explains (§ 4.2), and mitigates (§ 4.3) concept drift. Our main contributions are in the gray box marked with a star.

The explainer (section 4.2) is triggered at the time instances at which drift is detected. We design LEAF's explainer around the goal of characterizing errors of a regression model simply based on the model input and output. We extend global model-agnostic explanation methods [4, 5, 20] to identify and visualize the effect that different features have on prediction errors of black-box regression models. Doing so, the explainer determines representative features that contribute to drift, uses Local Error Approximation (LEA) to characterize the drift, and offer guidance for model compensation. The process generates LEAplots and LEAgrams, which provides insights for operators to understand drift events.

Based on the error distribution and statistical patterns, the mitigator (section 4.3) *automatically* forgets previous data, and performs informed replacement by sampling/over-sampling targeted regions. Our insight is that while the global error metrics provide a good measure of the performance over time, *the distribution of local errors across samples at each given time instance may be uneven.* Using this intuition, LEAF's mitigator better compensates for occurring drift.

4.2 Explaining Drift

The LEAF explanation module is based on the idea that in any regression-based black-box model, local errors may occur in a specific range of a given feature or correlated feature set. To assist mitigation and help operators understand drift, we define our goals of the drift explainer as follows: (1) to find and group features that contribute the most to drift, (2) to understand the extent of drift in a given range of values for feature(s), (3) to visualize errors in a spatio-temporal manner considering operational costs.

Multi-group correlated features. Natural correlations of features are often part of a dataset with a large number of features [25, 58]. These correlated features contribute to the performance of a model simultaneously. LEAF is designed to explain dynamic feature attributions by finding the representative features that provide the most valuable error approximations. To achieve this, we first rank features by permutation-based feature importance (*i.e.*, sensitivity score to permutation) [10]. Then, we group features by their correlations. The grouping stops when the feature has no importance value. Lastly, we choose the most representative (*i.e.*, highest importance score) feature from each group, as they can represent the statistical patterns (as well as error distribution) of the group. Using this technique, we obtain a set of representative features, denoted as $R(\rho)$, which indicates possible factors that lead to drift:

$$R(\rho) = \{\arg\max_{f_j \in g_i} F(f_j)\}_{j=1}^{n}, \tag{2}$$

where $G = \{g_i\}_{i=1}^{n}$ is the set of correlated feature groups such that for any features f_j, f_k in group g_i, $C(f_j, f_k) > \rho$. $F(f_i)$ represents the permutation-based feature importance for f_i.

Local error approximation (LEA). To analyze the error of a model on any corresponding dataset, we use the selected representative features R to inspect. For each feature f_i in R, we group samples based on the value of the representative feature into N bins (*i.e.*, quantiles $\{q_j\}_{j=1}^{N}$). The higher N is, the finer the granularity of local errors that can be observed. Next, a specified error metric (NRMSE by default) is computed for samples within each bin. We compile these N measurements into a vector, representing the error distribution. This technique approximates local errors over the range of the most sensitive and representative features for a specific model and dataset:

$$LEA(N, M) = \{\mathbf{e}_i | \mathbf{e}_{i,j} = Err(Q(f_i, q_j, (\mathbf{X}, \mathbf{y}))), f_i \in R\} \tag{3}$$

Here, $LEA(N, M)$ is the list of local error approximations for a given feature matrix $\mathbf{X}$, target vector $\mathbf{y}$ [4], number of bins N, and model M. Function $Q(f_i, q, (\mathbf{X}, \mathbf{y}))$ returns samples from quantile q_j of feature f_i, with $\mathbf{e}_i$ calculated for each representative feature from the set $R(\rho)$. LEA characterizes the extent of drift and focuses targeted drift mitigation. LEA and the feature grouping, provide the foundation of LEAF.

LEAplot and LEAgram. Starting from *LEA*, we develop LEAplot and LEAgram to assist LEAF users in visualizing local error components, comparing them across different data subsets, and better understanding which features most impact the occurrence of drift. For each feature from the set $R(\rho)$ and the concerned data subset, LEAplot shows the local errors in each bin. LEAplots show the different error distributions for representative features from different groups (see Figure 4).

LEAgrams, represented by LEAgram(T, N, M), augments LEAplot by incorporating temporal information along with error information. It shows the error for individual, binned samples arranged temporally. The test set is divided by time interval T (e.g., date) and assigns samples from those divided datasets into N bins, based on the quantiles of the most important feature f^* with regard to drift. When N is greater than or equal to the number of samples on each time interval, errors are shown for each sample.

In operational cellular networks, overestimation leads to different outcomes compared with underestimation when modeling for capacity planning purposes. For example, overestimation could result in unnecessary infrastructure expenditure, while underestimation can lead to user dissatisfaction as infrastructure is not augmented when it should be. Given these practical issues, we use Normalized Error (NE) as an error metric to preserve the sign in LEAgrams (see Figure 5).

4.3 Informed Mitigation

The mitigation module is based on the idea of informed adaptation. The key is to find the optimal data subset to retrain the model and mitigate drift. Given the information from LEAF's drift explainer, we can understand the features that contribute most to drift, the amount of drift at each range of values, and the over/underestimation status of each sample across time. We develop forgetting and over-sampling strategies using the information, and combine them organically based on dispersion (*i.e.*, Std/Mean in Table 1) of the target KPIs. We provide here a detailed description of the mitigation mechanisms, while an algorithmic description is provided in Appendix A.

Forgetting and over-sampling. When drift is detected, the latest set of samples is provided to the explanation module to derive the distribution of errors from LEA along with the values of the most important feature(s). For each feature f that contributes to drift, errors (NRMSE) are computed

[4]We suppress dependence on the data for ease of notation.

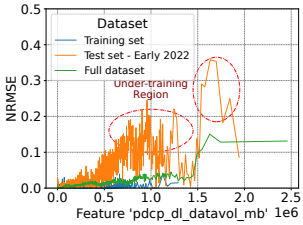
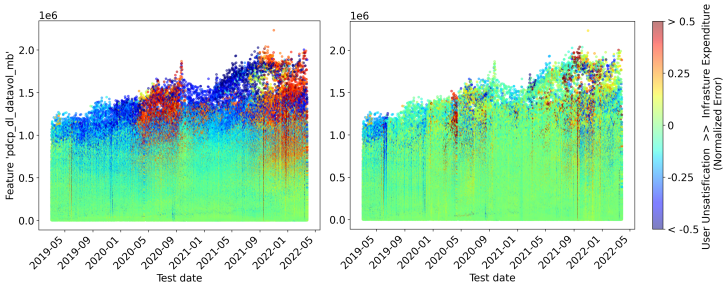


Fig. 4. The LEAplots (1,000 bins) that decomposes CatBoost NRMSE time-series. The distribution of estimated local error is shown along with the values of the most representative features `pdcp_dl_datavol_mb` (downlink volume in Mbps, Group 1).



(a) LEAgram (before mitigation). (b) LEAgram (after mitigation).

Fig. 5. The LEAgrams that decompose NRMSE time-series, where over- and under-estimation of models are different concerns of operators and they map back to extra infrastructure expenditure and lack of user satisfaction. (a) illustrates the decomposition of CatBoost NRMSE. (b) shows errors of mitigated CatBoost model if LEAF applied.

for N bins of the feature range, and they are associated with each bin. This error distribution LEA_{latest} shows, for a given feature, which ranges the model is highly erroneous or under-trained on. We first *forget* samples from the *previous training set* that cause high error. We assign weights to each sample from the previous training set based on the bin separations of LEA_{latest} that it falls into. The weight distribution W_{prev} is proportional to the area of the error. In those areas with high error contributions, we throw away the samples that contribute the most to the error (or equivalently, have lowest weight). We then over-sample from the latest data based on weights provided by LEA_{latest}, in order to pick samples that are the most representative of the current distribution. Now, retraining is triggered using this new training set. Moreover, the detection can be activated multiple times, so each round of forgetting and over-sampling is based on the previous round of the restructured training set.

KPI dispersion. The dispersion of a KPI (denoted as D_y) largely determines the appropriate mitigation approach and its aggression. If the dispersion of a KPI around the mean is high, it is more likely to be predicted wrong, and the forgetting and over-sampling strategy need to be more aggressive. We do grid search on the aggressiveness (linear, quadratic, or cubic weights, and dropping threshold) of the sample selection strategy for effective tuning. For example, for the target KPIs exhibiting D_y higher than 1 (*e.g.*, PU, CDR, GDR), we forget the samples of the original dataset with linear weights assigned in W_{prev}. In terms of over-sampling, we explicitly use non-linear (cubic) weights [31] of LEA_{latest} to focus on the regions of high error in the latest drifting instances. However, if D_y is low (*e.g.*, DVol, DTP, REst), it does not require a focused over-sampling because of a denser and more even feature space. We forget the samples of the original dataset with over 95% error, and use linearized weights of LEA_{latest} to over-sample the latest drifting instances. Note that we adjust the intensity of forgetting and oversampling to suit our problem. Interestingly, these fine-tuned parameters can be transferred from Fixed Dataset to Evolving Dataset, serving as a handy reference for tuning in other scenarios, depending on the characteristics of the KPIs.

5 CASE STUDY ON DRIFT EXPLANATION

In this section, we demonstrate how explanation tools of LEAF help operators to understand (1) what happens when a single instance of drift occurs (contributing factors and LEAplot); (2) an

end-to-end interpretation of a model or a mitigation scheme (LEAgram). Here, we showcase the above by applying *CatBoost to downlink volume forecasting*.

Contributing factors to drift. In early 2022, our drift detector alerts an instance of drift. It triggers the analysis of contributing features to understand what leads to drift (to verify if the drift explaining features carry the correct semantic meanings, we manually examine with operators). Each representative feature indicates a possible factor that leads to drift. The most representative feature of the first group is **pdcp_dl_datavol_mb**, the history of downlink volume itself, which serves as a sanity check. Highly correlated features include different stages of traffic communication: side channels that carry or reflect volume (*e.g.*, average downlink packets from UE), connection establishments and releases (*e.g.*, establishments of RRC), etc. All 32 features from this group jointly contribute to error, and thus will be used to explain the drift as a whole. Representative features from other groups provide different views of possible reasons. The second group contains features that reflect another reason: **badcoveragemeasurements**, which measures the bad coverage of eNodeBs in units of percent. This often arises from the geographical distance between an eNodeB and a UE, or destructive interference, which influences the downlink volume. The third representative feature is **rtp_gap_ratio_medium%**, which is related to specifically losing voice data packets during VoLTE call. With it being zero in most cases, bursty gaps between packets indeed affect traffic delivery.

Local error illustrations of drift using LEAplot. After the representative features are found, we apply LEA on them and illustrate the locally decomposed error of the drift by LEAplot per feature group, for a better understanding of the extent of drift within different value ranges of the feature. Operators can also localize the eNodeBs with bad performance. Figure 4 shows the LEAplot of the top representative feature. `pdcp_dl_datavol_mb` (the history of the downlink volume in Mb) is the most representative feature from Group 1 for this model. Although the training set has a low error when the feature value changes, when its value is between 0.6e6 and 1.3e6, errors in the "Early 2022" test set are more than 10x those of the training set and nearly 6x of those in the full test set. For the value above 1.5e6, the training set does not cover the range so the test sets observe high errors on it. The LEAplots of different representative features exhibit different distributions (see Appendix A), which can be used for multi-group mitigations. From LEAplot, we further trace back the eNodeBs that exhibit high errors. The top 5% of error mostly comes from eNodeBs located at suburban areas, because users there change their mobility pattern around this instance of drift, which is after the winter break.

End-to-end interpretation of performance using LEAgram. LEAgram helps operators derive an explanation for drift by revealing the changes of local errors and their durations over time. Operators can associate spatio-temporal changes of error with plausible reasons (*e.g.*, hardware installations, firmware updates, etc.) based on the location and duration of events. For this case, we interpret the error direction and extent before and after mitigations. Figure 5a shows the LEAgram. From March 15, 2020 to November 1, 2020, when the value of feature `pdcp_dl_datavol_mb` is above 1e6 Mb, large positive errors occur, implying overestimation. It indicates possible reasons as operators can correlate them with less user mobility, because people switch to broadband networks rather than cellular networks during the lockdown, thus less actual demands. Overestimations from this model occur again after October 2021, when the value of feature `pdcp_dl_datavol_mb` is above 0.6e6. If the operator were to base decisions on the output of this model, they may unnecessarily build new infrastructure. The model also has negative prediction errors above a value of 1e6, which could lead to user dissatisfaction as the operator may predict a lower demand than in reality. LEAgram also assists in demonstrating the performance of mitigation schemes over time. Figure 5b shows the mitigating effects of LEAF not only during COVID-19 lockdown, but also in early 2022 for gradual

shifts. Overall, 32.68% of reduction in $\Delta\overline{NRMSE}$ is shown, as compared to Figure 5a, with a major mitigation focus on the errors in the tail of the distribution.

6 EVALUATION

We now evaluate how the LEAF framework mitigates drift compared to existing techniques as baselines. We present how different mitigation schemes improve end-to-end model performance. We initially focus on the Fixed Dataset to provide an "apples to apples" comparison across models and to understand whether the inherently changing nature of the dataset lowers performance for some models. We then evaluate using the Evolving Dataset. Unless specified, we present the performance of LEAF with a single feature group.

6.1 End-to-End Comparison Across Mitigation Schemes

We compare the average NRMSE over the duration of the dataset against three baselines: ADaptive WINdowing (ADWIN) [9], which uses a variable-length window to detect and adjust for changes; the naïve retraining scheme (see Section 3.4) that retrains the model every 30 or 90 days (we choose these two frequencies because they present the best performance versus cost); and triggered retraining, which only utilizes the KSWIN detections, *i.e.*, retrain the model using the latest available data whenever drift is detected. To keep the evaluation fair, we use the same amount (14 days) of data for each retrain, which also controls the amount of time needed for a single retrain across schemes. The mitigation effectiveness is compared against a static model (trained on 14 days of data before July 1st, 2018) for each target KPI.

We investigate the trade-off between $\Delta\overline{NRMSE}$ and the number of retrains required for selected schemes and present in Table 3 a summary of the results (due to space limit, the full evaluation can be found in Table 6). Such trade-off provides insights not only on the performance of each mitigation scheme, but also on its applicability in practice in an operational network where each retrain operation might come at a cost. We use the number of retrains as a proxy to the system cost, as the operational overhead of implementing LEAF is minimal, especially when contrasted against the time-consuming process of retraining models. For example, running a CatBoost model on a 14-day training set averages around 113.688 seconds, whereas employing LEAF on the same volume of data takes merely 0.657 seconds—just about 0.578% of the time needed for CatBoost. Our goal is to find the scheme that achieves the best mitigation effectiveness first, while balancing the alternative goal of few retrains.

Despite tuning its confidence parameters and providing sufficient window size, ADWIN struggles to effectively mitigate NMRSE induced by drift. For CatBoost, it is only triggered 7 times, achieving a −9.75% mitigation effectiveness on DVol predictions. It even fails to detect any drift in the error time series of highly dispersed KPIs like CDR and GDR. This is largely due to the cyclical patterns and noise irregularities inherent in our real-world data streams.

Naïve retraining every 30 days requires the highest number (39 for all KPIs) of retrains, yet its mitigation effectiveness never outperforms LEAF. Naïve retraining every 90 days requires fewer retrains (13). However, the mitigation effectiveness is frequently inferior to LEAF's, except for the CDR KPI. The triggered mitigation scheme rarely outperforms other schemes on either metric. It also has exponential errors for KPIs like GDR (44.56% increase on $\Delta\overline{NRMSE}$), making it less practical among the schemes since it does not guarantee performance improvements after mitigation.

Finally, LEAF consistently outperforms all baseline schemes considering the trade-off. Due to its error explanation and informed mitigation approach, LEAF already exceeds the performance of other methods across KPIs (*e.g.*,−46.69% for PU, −38.44% for REst), except for CDR, even when only using one representative feature. With additional feature groups, it can achieve either greater

Table 3. Effectiveness of mitigation schemes measured in $\Delta\overline{NRMSE}$ and *#Retrains* (both are the lower the better) using Fixed Dataset. We include models from different model families over a variety of KPIs. The scheme with the highest performance is shaded in gray.

Model	KPIs	$\Delta\overline{NRMSE}$ (#Retrains) of Mitigation Schemes					
		Naïve$_{30}$		Triggered		LEAF	
CatBoost	DVol	−29.62%	(39)	−31.80%	(27)	−32.67%	(28)
	PU	−44.88%	(39)	−35.06%	(25)	−46.59%	(35)
	DTP	−20.02%	(39)	−23.84%	(28)	−24.30%	(31)
	REst	−35.41%	(39)	−38.38%	(25)	−38.44%	(31)
	CDR	2.35%	(39)	−4.21%	(17)	−3.63%	(9)
	GDR	3.37%	(39)	44.56%	(17)	−6.24%	(19)
ExtraTrees	DVol	−24.77%	(39)	−28.17%	(32)	−30.64%	(32)
	PU	−44.26%	(39)	−50.76%	(26)	−45.83%	(27)
	DTP	−18.13%	(39)	−21.63%	(32)	−22.59%	(23)
	REst	−31.95%	(39)	−34.29%	(22)	−36.13%	(29)
	CDR	2.10%	(39)	8.08%	(20)	−0.20%	(11)
	GDR	−0.58%	(39)	33.67%	(17)	−14.26%	(19)
LSTM	DVol	0.54%	(39)	14.12%	(21)	2.67%	(19)
	PU	37.11%	(39)	3.76%	(25)	−20.48%	(18)
	DTP	17.08%	(39)	−0.82%	(14)	−37.13%	(20)
	REst	6.78%	(39)	5.78%	(27)	4.21%	(26)
	CDR	−33.22%	(39)	−21.39%	(10)	−71.52%	(11)
	GDR	0.41%	(39)	−8.58%	(14)	−16.29%	(13)
KNeighbors	DVol	−8.26%	(39)	−4.11%	(16)	−4.47%	(24)
	PU	−34.09%	(39)	−37.99%	(16)	−18.11%	(20)
	DTP	−4.73%	(39)	−4.03%	(18)	−1.53%	(22)
	REst	−26.69%	(39)	−25.86%	(25)	−22.10%	(16)
	CDR	9.44%	(39)	7.35%	(11)	4.69%	(12)
	GDR	−8.13%	(39)	−23.40%	(19)	−6.12%	(13)

effectiveness or fewer retrains. Compared to triggered retraining, LEAF can mitigate more errors, with a similar or slightly higher number of retrains (*e.g.*, all but CDR). For CDR, LEAF is able to mitigate a similar amount of errors with 30.8% fewer retrains. The number of feature groups influences LEAF's effectiveness, which we will discuss in more detail later.

6.2 Sensitivity Analysis

In this section, we examine the configuration sensitivity of LEAF, considering the number of feature groups used, and its performance across models, KPIs, and morphologies.

Different number of feature groups. Multi-group LEAF forgets and over-samples the input dataset according to the error distributions from more than one representative feature. Although the amount of data being resampled remains the same, multi-group LEAF iteratively optimizes which retrain data to use based on the input of the drift explanation component. As shown in Table 5 in Appendix B, the errors that are mitigated tend to increase when more feature groups are used, except for GDR. 0.34% to 2.83% more errors are mitigated by multi-group LEAF compared to single-group LEAF. However, the optimal number of feature groups is not constant across KPIs. Although multi-round LEAF is better than single-round LEAF in most cases, it does not show continued improvement when we add more features. For DVol (−35.12% of $\Delta\overline{NRMSE}$, 34 retrains), PU (−47.62%, 27), and REst (−41.27%, 32), three feature groups lead to the highest mitigation

Table 4. Effectiveness measured in $\Delta\overline{NRMSE}$ for different morphologies (DVol predicted by CatBoost).

Schemes	$\Delta\overline{NRMSE}$ of Different Morphologies			
	All	Rural	Suburban	Urban
ADWIN	−9.75%	−10.01%	−13.26%	−14.24%
Naïve_{30}	−29.62%	−27.21%	−31.76%	−33.86%
Naïve_{90}	−19.83%	−17.19%	−21.17%	−27.83%
Triggered	−31.80%	−27.32%	−34.98%	−2.91%
LEAF	−32.67%	−31.29%	−36.60%	−33.66%

effectiveness. For DTP (−24.64%, 30) and CDR (−6.22%, 12), five feature groups work best. As the most dispersed KPI, GDR (−6.24%, 19) is an outlier that reaches optimality with only one feature group. Feature importance and the number of features within each group might be the factors that affect the effectiveness of the mitigation.

Different models. According to Table 3, as expected, we find that each model responds differently to the mitigation schemes. LEAF outperforms baselines across most combinations of models and KPIs, with the exception of KNeighbors. For CatBoost and ExtraTrees, LEAF is either the most effective or very close to the best performing scheme across all KPIs. Further, LEAF consistently mitigates drift across all models, *i.e.*, their $\Delta\overline{NRMSE}$s are always negative. Other schemes, like naïve retraining or triggered retraining, do not uniformly achieve improvement across KPIs. For example, for CDR and GDR, both schemes end increasing errors after mitigation up to 44.56% in comparison to the static models.

Additionally to improving model performance, LEAF achieves the best results while requiring 10.3% to 76.9% fewer retrains for CatBoost and 17% to 71.8% fewer for ExtraTrees when compared to naïve retraining every 30 days. This pattern persists across other boosting (*e.g.*, LightGBM) and bagging (*e.g.*, Random Forest) algorithms.

For LSTM, we find that LEAF is drastically better at reducing NRMSE. Despite achieving slightly worse performance for DVol and REst, LEAF is the most effective scheme for the other KPIs. Moreover, when effective, LEAF reduces the NRMSE by a large margin compared with the second best results. By applying LEAF, LSTM achieves 7.71% to 50.13% less NRMSE than triggered retraining. Surprisingly, CDR errors are reduced by 71.52% when only 11 retrains are required.

LEAF is less effective with KNeighbors and other distance-based models, likely due to their unique expressiveness and generalization [35] approach. KNeighbors uses a lazy regressor, memorizing the entire training set for predictions based on nearest neighbor distance. Over-sampling error regions may lead to poorer performance in other areas due to unbalanced sample addition. Conversely, bagging methods train independent learners in parallel [23] (or sequentially with boosting) [17]), resulting in less impact from previous learners during targeted mitigation.

Different KPIs. Different KPI time-series respond variably to mitigation depending on their unique characteristics. For instance, PU, CDR, and GDR are among the toughest KPIs to mitigate in CatBoost and ExtraTrees, with PU having the highest NMRSE. This stems from PU's propensity for sudden data losses, causing high variability and model collapse. Meanwhile, despite lower baseline NMRSEs, CDR and GDR are challenging due to high-frequency variation. This is confirmed by their high coefficients of variance - 1.34, 1.35, and 2.12 respectively. Methods like triggered retraining escalate NMRSE for these KPIs. Conversely, less dispersed KPIs like DVol, DTP, and REst, despite their drift, are more adaptive due to homogeneous distribution changes.

Different Morphologies. We divide Fixed Dataset into subnets based on geographical categories: rural, suburban, and urban. Our findings, detailed in Table 4, show that the performance improvement, measured in $\Delta\overline{NRMSE}$, is most pronounced in urban areas, followed by suburban and rural areas. The LEAF scheme consistently delivers good results across all areas, while the Triggered scheme underperforms in urban settings, possibly due to high variance of features in this setting.

These insights are crucial for mitigating drift as networks transition from 4G LTE to 5G and beyond. Given the increased use of smaller cells in 5G networks, there's a need for schemes that are effective for both microcells deployed in high-density urban settings as well as macrocells deployed in rural and suburban regions. Our study suggests that schemes like LEAF could be suitable for deployments in all areas, regardless of RF environment.

6.3 LEAF Effectiveness on Evolving Infrastructure

In the previous analysis, we focused on a fixed number of eNBs in the Fixed Dataset to evaluate internal drift factors like software upgrades and user behavior pattern changes. Here, we use the Evolving Dataset to test the influence of daily sample numbers and changing infrastructure, as new eNodeBs are being constantly deployed.

We evaluate mitigation schemes across datasets; Table 7 in Appendix B shows results in more detail, where we show the best multi-group LEAF scheme, denoted as LEAF*. Both naïve retraining schemes perform similarly across most KPIs, except for CDR and GDR with a 30-day retraining frequency. This improvement is likely due to the high retraining frequency capturing new eNodeBs. This trend is also seen with triggered retraining, which performs better on the Evolving Dataset across most KPIs, highlighting the benefits of a timely drift detector. LEAF and LEAF* consistently outperform other strategies (*e.g.*, LEAF* achieves −48.01% mitigation effectiveness compared to Triggered −44.01% for REst in Evolving Dataset), demonstrating the advantages of integrating a drift detector and a targeted retraining strategy based on multi-group feature error information.

7 RELATED WORK

Drift in network management. Machine learning has been applied to many networking problems, including anomaly detection [52, 53], intrusion detection [16, 54], cognitive network management [6], and network forecasting [13, 32, 44]. It is well-known that network traffic is inherently variable over time, making it challenging to deploy machine learning [55]. For example, features such as latency can also drift over long periods of time [46]. Moreover, the COVID-19 pandemic has provided a unique example of sudden drift and its effects on network traffic patterns, network usage, and resulting model accuracy [15, 19, 34, 37, 43]. A previous study [14] proposes an ensemble learning method to detect concept drift in cellular networks, but not to mitigate or explain it. Also, due to LEAF's effectiveness across a diverse set of KPIs with drastically different behavior over time, we believe it can be applied in other problems. We leave this investigation to future work.

Change detection and impact assessment. A growing body of research focuses on the development of tools that employ advanced algorithms to rapidly and robustly detect changes, such as software or firmware upgrades and configuration alterations, in operational networks [40, 41] and internet-based services [39, 63], while efficiently assessing their impact on network performance. MERCURY [41] identifies network performance changes using statistical rule mining and network configuration analysis, while Litmus [40] evaluates the impact of these changes, comparing performance between study and control groups via a spatial regression algorithm. For internet-based services, PRISM [39] and FUNNEL [63] utilize association between maintenance and network elements and a difference-in-difference method to assess the impact. LEAF, in contrast, focuses on

addressing drift in ML models and explains and mitgates drift end-to-end. The explanations do not require external knowledge, utilizing black-box models instead of white-box logs.

Explainable AI and drift explanation. Explainable AI has recently attempted to address the challenge of black-box model interpretation [3, 26, 48]. Global model-agnostic methods, such as Partial Dependence Plot [20] and Accumulated Local Effects plot [4, 5], provide visual clues on the effect of different features on the prediction of black-box models. LEAF's LEAplot and LEAgram are inspired by PDP and ALE, but extend these techniques by (1) showing errors instead of effects; (2) identifying the correlated sets of features that contribute to drift, and (3) helping to visualize how drift evolves over time in operation. Transcend [27] uses statistical comparison of samples to create decision boundary-based explanation. CADE [61] uses contrastive learning to develop a distance-based explanation to find the features that have the largest distance changes towards the centroid in the latent space. These methods, however, only apply to classification problems.

Drift mitigation. Adapting a model to mitigate concept drift is a well-explored area [22, 36], but only a few approaches surpass frequent retraining [28, 42, 56, 62]. Adaptation methods can be based on retraining [60] or model ensembles [30]. Retraining-based approaches often necessitate complex data management, significant storage, and memory. Paired Learners [7] utilize one stable learner for old data and another reactive learner for the latest data. Accuracy Updated Ensemble (AUE2)[11, 12] incrementally updates sub-models on a small portion of data to adapt to drift. For software system anomaly detection, the framework StepWise[38] detects concept drift, takes external factors into account to distinguish expected and unexpected drift, and enables rapid adaptation by any anomaly detection algorithm. However, their mitigation heavily relies on the assumption that old and new concepts lie on a linear manifold, while LEAF makes no such assumptions.

8 CONCLUSION

An important step in deploying machine learning models for networking tasks in practice is dealing with concept drift. Although this phenomenon has been explored in other contexts, it has received limited attention in the networking domain. To address it, this paper characterizes drift patterns across multiple models and KPIs and has developed, presented, and evaluated LEAF, a framework to detect, explain, and mitigate drift for black-box models applied to cellular demand forecasting. The LEAF framework employs explainable AI and informed mitigation. Our results based on more than four years of KPI data from this large cellular network show that LEAF consistently outperforms both periodic and triggered retraining, while reducing the cost of retraining. The LEAF framework can likely also be applied to other network management problems modeled as regression-based predictions. Another area for exploration lies in the use of ensemble models across modeling frameworks in LEAF, as they could potentially enhance resilience against concept drift. Finally, the evaluation of LEAF to date has been conducted on fully labeled datasets, due to the forecasting problem nature, where the ground truth is available once the future is observed; thus, a promising direction could be to improve LEAF to cope with semi-supervised or unsupervised regressors.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive feedback. This research is supported by grants from NSF CNS-2028145, CNS-2124393 and CNS-2126327, as well as ANR Project No ANR-21-CE94-0001-01 (MINT) and the France and Chicago Collaborating in the Sciences program.

REFERENCES

[1] 2023. LTE cellular network performance indicators daily measurements dataset. https://forms.gle/g5pbB5qRHeBsEmZJ6.

[2] AutoGluon AI. accessed July, 2021. *AutoGluon: AutoML for Text, Image, and Tabular Data.* https://auto.gluon.ai/stable/index.html.

[3] David Alvarez-Melis and Tommi S Jaakkola. 2018. On the robustness of interpretability methods. *arXiv preprint arXiv:1806.08049* (2018).

[4] Daniel W Apley and Jingyu Zhu. 2020. Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 82, 4 (2020), 1059–1086.

[5] Behnaz Arzani, Kevin Hsieh, and Haoxian Chen. 2021. Interpretable Feedback for AutoML and a Proposal for Domain-customized AutoML for Networking. In *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks*. 53–60.

[6] Sara Ayoubi, Noura Limam, Mohammad A Salahuddin, Nashid Shahriar, Raouf Boutaba, Felipe Estrada-Solano, and Oscar M Caicedo. 2018. Machine learning for cognitive network management. *IEEE Communications Magazine* 56, 1 (2018), 158–165.

[7] Stephen H Bach and Marcus A Maloof. 2008. Paired learners for concept drift. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 23–32.

[8] Roberto Souto Maior Barros and Silas Garrido T Carvalho Santos. 2018. A large-scale comparison of concept drift detectors. *Information Sciences* 451 (2018), 348–370.

[9] Albert Bifet and Ricard Gavalda. 2007. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*. SIAM, 443–448.

[10] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[11] Dariusz Brzeziński and Jerzy Stefanowski. 2011. Accuracy updated ensemble for data streams with concept drift. In *International conference on hybrid artificial intelligence systems*. Springer, 155–163.

[12] Dariusz Brzezinski and Jerzy Stefanowski. 2013. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Transactions on Neural Networks and Learning Systems* 25, 1 (2013), 81–94.

[13] Sandeep Chinchali, Pan Hu, Tianshu Chu, Manu Sharma, Manu Bansal, Rakesh Misra, Marco Pavone, and Sachin Katti. 2018. Cellular network traffic scheduling with deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*.

[14] Gabriela F Ciocarlie, Ulf Lindqvist, Szabolcs Nováczki, and Henning Sanneck. 2013. Detecting anomalies in cellular networks using an ensemble method. In *Proceedings of the 9th international conference on network and service management (CNSM 2013)*. IEEE, 171–174.

[15] Comcast. accessed October, 2020. *COVID-19 Network Update.* https://corporate.comcast.com/covid-19/network/may-20-2020.

[16] Bo Dong and Xue Wang. 2016. Comparison deep learning method to traditional methods using for network intrusion detection. In *2016 8th IEEE International Conference on Communication Software and Networks (ICCSN)*. IEEE, 581–585.

[17] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. 2018. CatBoost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363* (2018).

[18] Florentino Fdez-Riverola, Eva Lorenzo Iglesias, Fernando Díaz, José Ramon Méndez, and Juan M Corchado. 2007. Applying lazy learning algorithms to tackle concept drift in spam filtering. *Expert Systems with Applications* 33, 1 (2007), 36–48.

[19] Anja Feldmann, Oliver Gasser, Franziska Lichtblau, Enric Pujol, Ingmar Poese, Christoph Dietzel, Daniel Wagner, Matthias Wichtlhuber, Juan Tapidor, Narseo Vallina-Rodriguez, et al. 2020. The Lockdown Effect: Implications of the COVID-19 Pandemic on Internet Traffic.. In *Internet Measurement Conference (IMC '20)*.

[20] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[21] Futuriom. accessed August, 2021. *Verizon Applies Machine Learning to Operations.* https://www.futuriom.com/articles/news/verizon-applies-machine-learning-to-operations/2018/08.

[22] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys (CSUR)* 46, 4 (2014), 1–37.

[23] Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely randomized trees. *Machine learning* 63, 1 (2006), 3–42.

[24] Paulo M Gonçalves Jr, Silas GT de Carvalho Santos, Roberto SM Barros, and Davi CL Vieira. 2014. A comparative study on concept drift detectors. *Expert Systems with Applications* 41, 18 (2014), 8144–8156.

[25] Baptiste Gregorutti, Bertrand Michel, and Philippe Saint-Pierre. 2017. Correlation and variable importance in random forests. *Statistics and Computing* 27, 3 (2017), 659–678.

[26] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. 2018. Lemna: Explaining deep learning based security applications. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 364–379.

[27] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting concept drift in malware classification models. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 625–642.

[28] Alex Kantchelian, Sadia Afroz, Ling Huang, Aylin Caliskan Islam, Brad Miller, Michael Carl Tschantz, Rachel Greenstadt, Anthony D Joseph, and JD Tygar. 2013. Approaches to adversarial drift. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*. 99–110.

[29] Matthew Brian Knebl and Amr Albanna. 2016. Systems and methods for network performance forecasting. US Patent 9,439,081.

[30] Bartosz Krawczyk, Leandro L Minku, Joao Gama, Jerzy Stefanowski, and Michał Woźniak. 2017. Ensemble learning for data stream analysis: A survey. *Information Fusion* 37 (2017), 132–156.

[31] SK Lahiri and SN Lahiri. 2003. *Resampling methods for dependent data*. Springer Science & Business Media.

[32] Shinan Liu, Francesco Bronzino, Paul Schmitt, Nick Feamster, Ricardo Borges, Hector Garcia Crespo, and Brian Ward. 2021. Understanding Model Drift in a Large Cellular Network. *CoRR* (2021).

[33] Shinan Liu, Tarun Mangla, Ted Shaowang, Jinjin Zhao, John Paparrizos, Sanjay Krishnan, and Nick Feamster. 2023. AMIR: Active Multimodal Interaction Recognition from Video and Network Traffic in Connected Environments. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 7, 1 (2023). https://doi.org/10.1145/3580818

[34] Shinan Liu, Paul Schmitt, Francesco Bronzino, and Nick Feamster. 2021. Characterizing Service Provider Response to the COVID-19 Pandemic in the United States. In *PAM 2021-Passive and Active Measurement Conference*.

[35] Viktor Losing, Barbara Hammer, and Heiko Wersing. 2016. KNN classifier with self adjusting memory for heterogeneous concept drift. In *2016 IEEE 16th international conference on data mining (ICDM)*. IEEE, 291–300.

[36] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. 2018. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering* 31, 12 (2018), 2346–2363.

[37] Andra Lutu, Diego Perino, Marcelo Bagnulo, Enrique Frias-Martinez, and Javad Khangosstar. 2020. A characterization of the covid-19 pandemic impact on a mobile network operator traffic. In *Proceedings of the ACM Internet Measurement Conference*. 19–33.

[38] Minghua Ma, Shenglin Zhang, Dan Pei, Xin Huang, and Hongwei Dai. 2018. Robust and rapid adaption for concept drift in software system anomaly detection. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 13–24.

[39] Ajay Mahimkar, Zihui Ge, Jia Wang, Jennifer Yates, Yin Zhang, Joanne Emmons, Brian Huntley, and Mark Stockert. 2011. Rapid detection of maintenance induced changes in service performance. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*. 1–12.

[40] Ajay Mahimkar, Zihui Ge, Jennifer Yates, Chris Hristov, Vincent Cordaro, Shane Smith, Jing Xu, and Mark Stockert. 2013. Robust assessment of changes in cellular networks. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. 175–186.

[41] Ajay Anil Mahimkar, Han Hee Song, Zihui Ge, Aman Shaikh, Jia Wang, Jennifer Yates, Yin Zhang, and Joanne Emmons. 2010. Detecting the performance impact of upgrades in large operational networks. In *Proceedings of the ACM SIGCOMM 2010 Conference*. 303–314.

[42] Ankur Mallick, Kevin Hsieh, Behnaz Arzani, and Gauri Joshi. 2022. Matchmaker: Data Drift Mitigation in Machine Learning for Large-Scale Systems. *Proceedings of Machine Learning and Systems* 4 (2022), 77–94.

[43] Martin McKeay. accessed October, 2020. *Parts of a Whole: Effect of COVID-19 on US Internet Traffic*. https://blogs.akamai.com/sitr/2020/04/parts-of-a-whole-effect-of-covid-19-on-us-internet-traffic.html.

[44] Lifan Mei, Runchen Hu, Houwei Cao, Yong Liu, Zifan Han, Feng Li, and Jin Li. 2020. Realtime mobile bandwidth prediction using LSTM neural network and Bayesian fusion. *Computer Networks* 182 (2020), 107515.

[45] Robert Nau. 2014. Notes on linear regression analysis. *Fuqua School of Business, Duke University Retrieved from: https://people.duke.edu/~rnau/Notes_on_linear_regression_analysis--Robert_Nau.pdf* (2014).

[46] Ashkan Nikravesh, David R Choffnes, Ethan Katz-Bassett, Z Morley Mao, and Matt Welsh. 2014. Mobile network performance from user devices: A longitudinal, multidimensional analysis. In *International Conference on Passive and Active Network Measurement*. Springer, 12–22.

[47] Christoph Raab, Moritz Heusinger, and Frank-Michael Schleif. 2020. Reactive soft prototype computing for concept drift streams. *Neurocomputing* 416 (2020), 340–351.

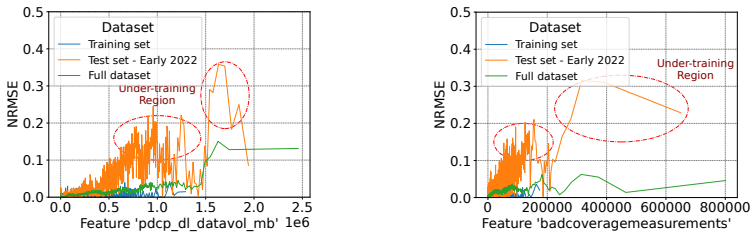
[48] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1135–1144.

[49] Janne Riihijarvi and Petri Mahonen. 2018. Machine learning for performance prediction in mobile cellular networks. *IEEE Computational Intelligence Magazine* 13, 1 (2018), 51–60.
[50] Jeffrey C Schlimmer and Richard H Granger. 1986. Incremental learning from noisy data. *Machine learning* 1, 3 (1986), 317–354.
[51] Joan Serra, Ilias Leontiadis, Alexandros Karatzoglou, and Konstantina Papagiannaki. 2017. Hot or not? Forecasting cellular network hot spots using sector performance indicators. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 259–270.
[52] Taeshik Shon, Yongdae Kim, Cheolwon Lee, and Jongsub Moon. 2005. A machine learning framework for network anomaly detection using SVM and GA. In *Proceedings from the sixth annual IEEE SMC information assurance workshop*. IEEE, 176–183.
[53] Taeshik Shon and Jongsub Moon. 2007. A hybrid machine learning approach to network anomaly detection. *Information Sciences* 177, 18 (2007), 3799–3821.
[54] Chris Sinclair, Lyn Pierce, and Sara Matzner. 1999. An application of machine learning to network intrusion detection. In *Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99)*. IEEE, 371–377.
[55] Robin Sommer and Vern Paxson. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*. IEEE, 305–316.
[56] Kurt Thomas, Chris Grier, Justin Ma, Vern Paxson, and Dawn Song. 2011. Design and evaluation of a real-time url spam filtering service. In *2011 IEEE symposium on security and privacy*. IEEE, 447–462.
[57] Maurras Ulbricht Togbe, Yousra Chabchoub, Aliou Boly, Mariam Barry, Raja Chiky, and Maroua Bahri. 2021. Anomalies Detection Using Isolation in Concept-Drifting Data Streams. *Computers* 10, 1 (2021), 13.
[58] Laura Toloşi and Thomas Lengauer. 2011. Classification with correlated features: unreliability of feature ranking and solutions. *Bioinformatics* 27, 14 (2011), 1986–1994.
[59] Qiang Xu, Sanjeev Mehrotra, Zhuoqing Mao, and Jin Li. 2013. PROTEUS: network performance forecast for real-time, interactive mobile applications. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. 347–360.
[60] Shuliang Xu and Junhong Wang. 2017. Dynamic extreme learning machine for data stream classification. *Neurocomputing* 238 (2017), 433–449.
[61] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. 2021. {CADE}: Detecting and Explaining Concept Drift Samples for Security Applications. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*.
[62] Xiaoyu You, Mi Zhang, Daizong Ding, Fuli Feng, and Yuanmin Huang. 2021. Learning to learn the future: Modeling concept drifts in time series prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2434–2443.
[63] Shenglin Zhang, Ying Liu, Dan Pei, Yu Chen, Xianping Qu, Shimin Tao, and Zhi Zang. 2015. Rapid and robust impact assessment of software changes in large internet-based services. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. 1–13.
[64] Gan Zheng, Ioannis Krikidis, Christos Masouros, Stelios Timotheou, Dimitris-Alexandros Toumpakaris, and Zhiguo Ding. 2014. Rethinking the role of interference in wireless networks. *IEEE Communications Magazine* 52, 11 (2014), 152–158.
[65] Indrė Žliobaitė. 2010. Adaptive training set formation. (2010).

A LEAF DETAILS

Fixed Dataset. We provide the details of characteristics of target KPIs in Fixed Dataset here. The dispersions (Std/Mean) of KPIs are DVol (0.73) & PU (1.34), DTP (0.57) & REst(0.77), CDR (1.35) & GDR (2.12). Still, it covers a variety of statistical patterns, which shows the diversity of this real-world dataset. Compared to Evolving Dataset, the dispersions (Std/Mean) of KPIs are less because no new sites are added.

LEAF detector. The drift detection module monitors the NRMSE time-series and notifies operators when drift occurs. We feed NRMSE time-series into Kolmogorov-Smirnov Windowing (KSWIN) [47, 57]. KSWIN is a recent concept drift detection method [47] based on KS test, which is a non-parametric statistical test that makes no assumption of underlying data distributions [57]. It monitors the performance distributions as data arrives, using CatBoost NRMSE time-series to forecast downlink volume as an example. The model is trained on 14 days of data before July 1, 2018 (Figure 2a). Without ground truths of drifts in our dataset, we cross-check detection results with clear signals like missing data and COVID-19 network changes. KSWIN detects drift instances during major anomalies (around June 2019, December 2019, and April 2021) and the COVID-19 quarantine period. Tested across NRMSE time-series of five KPIs, using different models and training set sizes and periods, KSWIN consistently performs well.



(a) Feature group 1.

(b) Feature group 2.

Fig. 6. The LEAplots (1,000 bins) that decomposes CatBoost NRMSE time-series. The distribution of estimated local error is shown along with the values of the most representative features `pdcp_dl_datavol_mb` (downlink volume in Mbps, Group 1) and `badcoveragemeansurements` (number of bad coverages, Group 2).

LEAplot for different feature groups. Figure 6 shows the LEAplots of representative features from the top two feature groups of our case study. In Figure 6a, `pdcp_dl_datavol_mb` (the history of downlink volume) is the most representative feature from Group 1 for this model. Although the training set has a very low error when the feature value changes, when its value is between 0.6e6 and 1.3e6, the errors in "Early 2022" test set are more than 10x those of the training set and nearly 6x of those in the full test set. For the value above 1.5e6, the training set does not cover the range so the test sets observe high errors on it. Figure 6b, however, shows the LEAplots of representative feature `badcoveragemeasurements` from Group 2. When the value is above 2e5, "Early 2022" test set has more errors than training set, mainly because that the training set does not cover the range, and the model is poorly fitted there.

These two LEAplots exhibit very different error distributions. For example, samples from the high error region of Figure 6a are not always in that of Figure 6b. It indicates that we can leverage this difference for iterative mitigation on different feature groups.

Informed mitigation algorithmic representation. Algorithm 1 shows the mitigation module procedure. When drift is detected, the latest samples are provided to the explanation module to derive the distribution of errors from LEA along with the values of the most important feature(s). If no mitigation has occurred, $\mathbf{X}_{\text{new}}$ is the original dataset.

Algorithm 1 Informed Mitigation

Require: $\mathbf{X}_{\text{new}}$: previous training feature matrix, $\mathbf{X}_{\text{latest}}$: latest feature matrix, $\mathbf{y}_{\text{new}}$: previous training target vector, $\mathbf{y}_{\text{latest}}$: latest target vector, M_{prev}: the previous model

Compute KPI dispersion D_y based on y_{prev}
for $t \in \{1, \ldots, T\}$ **do**
  if KSWIN returns DriftDetected == True **then**
    Set $(\mathbf{X}_{\text{prev}}, \mathbf{y}_{\text{prev}}) = (\mathbf{X}_{\text{new}}, \mathbf{y}_{\text{new}})$
    Initialize new training set $(\mathbf{X}_{\text{new}}, \mathbf{y}_{\text{new}}) = (\emptyset, \emptyset)$
    for each $f \in R(\mathbf{X}_{\text{prev}}, \mathbf{y}_{\text{prev}}, \rho)$ **do**
      Compute $LEA_{\text{latest}} = LEA(\mathbf{X}_{\text{latest}}, \mathbf{y}_{\text{latest}}, N, M_{\text{prev}}, f)$
      //**Forgetting**
      Initialize W_{prev} as $\emptyset$
      for each sample $(\mathbf{x}, y)$ in $\mathbf{X}_{\text{prev}}, \mathbf{y}_{\text{prev}}$ **do**
        Find bin b of LEA_{latest} that $(\mathbf{x}, y)$ belongs to
        Assign weight w to $(\mathbf{x}, y)$ based on LEA_{latest}, b
        Add w to W_{prev}
      end for
      Delete samples from $(\mathbf{X}_{\text{prev}}, \mathbf{y}_{\text{prev}})$ based on D_y and W_{prev}
      Add remaining samples to $(\mathbf{X}_{\text{new}}, \mathbf{y}_{\text{new}})$
      //**Over-sampling**
      Sample $\mathbf{X}_{\text{latest}}, \mathbf{y}_{\text{latest}}$ based on D_y and LEA_{latest}
      Add $\mathbf{X}'_{\text{latest}}, \mathbf{y}'_{\text{latest}}$ to $(\mathbf{X}_{\text{new}}, \mathbf{y}_{\text{new}})$
    end for
    Retrain M_{new} using $(\mathbf{X}_{\text{new}}, \mathbf{y}_{\text{new}})$
  end if
end for

B SUPPLEMENTARY RESULTS

Supplementary results showcase multi-group LEAF's effectiveness across CatBoost KPIs (Table 5), and comprehensive evaluations across all KPIs, models, and mitigation schemes. Tables 6 and 7 present full evaluations of all mitigation schemes' effectiveness, measured using Fixed Dataset and Evolving Dataset respectively.

Table 5. Effectiveness of mitigation schemes, measured using CatBoost on Fixed Dataset, is presented; lower values are better. The number near LEAF indicates the number of feature groups considered.

Mitigation	$\Delta\overline{NRMSE}$ **(#Retrains) for KPIs**											
Schemes	DVol		PU		DTP		REst		CDR		GDR	
LEAF_1	−32.67%	(28)	−46.59%	(35)	−24.30%	(31)	−38.44%	(31)	−3.63%	(9)	−6.24%	(19)
LEAF_3	−35.12%	(34)	−47.62%	(27)	−23.38%	(26)	−41.27%	(32)	−3.91%	(7)	24.62%	(17)
LEAF_5	−33.28%	(26)	−45.85%	(27)	−24.64%	(30)	−38.07%	(29)	−6.22%	(12)	6.41%	(18)

Received November 2022; revised May 2023; accepted June 2023

Table 6. Effectiveness of mitigation schemes measured in $\Delta\overline{NRMSE}$ and *#Retrains* (both are the lower the better) using Fixed Dataset. We include representative models from different model families over a variety of KPIs.

Model	KPIs	$\Delta\overline{NRMSE}$ (#Retrains) of Mitigation Schemes									
		ADWIN		Naïve_{30}		Naïve_{90}		Triggered		LEAF	
CatBoost	DVol	−9.75%	(7)	−29.62%	(39)	−19.83%	(13)	−31.80%	(27)	−32.67%	(28)
	PU	−29.76%	(7)	−44.88%	(39)	−16.44%	(13)	−35.06%	(25)	−46.59%	(35)
	DTP	−1.16%	(4)	−20.02%	(39)	−16.88%	(13)	−23.84%	(28)	−24.30%	(31)
	REst	−16.01%	(8)	−35.41%	(39)	−26.25%	(13)	−38.38%	(25)	−38.44%	(31)
	CDR	0%	(0)	2.35%	(39)	−5.39%	(13)	−4.21%	(17)	−3.63%	(9)
	GDR	0%	(0)	3.37%	(39)	−4.20%	(13)	44.56%	(17)	−6.24%	(19)
ExtraTrees	DVol	−7.86%	(6)	−24.77%	(39)	−15.10%	(13)	−28.17%	(32)	−30.64%	(32)
	PU	−28.43%	(7)	−44.26%	(39)	−16.30%	(13)	−50.76%	(26)	−45.83%	(27)
	DTP	−0.01%	(3)	−18.13%	(39)	−13.80%	(13)	−21.63%	(32)	−22.59%	(23)
	REst	−11.42%	(7)	−31.95%	(39)	−22.28%	(13)	−34.29%	(22)	−36.13%	(29)
	CDR	0%	(0)	2.10%	(39)	−2.52%	(13)	8.08%	(20)	−0.20%	(11)
	GDR	0%	(0)	−0.58%	(39)	9.64%	(13)	33.67%	(17)	−14.26%	(19)
LSTM	DVol	7.23%	(6)	0.54%	(39)	−0.97%	(13)	14.12%	(21)	2.67%	(19)
	PU	−4.57%	(6)	37.11%	(39)	12.65%	(13)	3.76%	(25)	−20.48%	(18)
	DTP	−6.29%	(4)	17.08%	(39)	11.94%	(13)	−0.82%	(14)	−37.13%	(20)
	REst	−11.90%	(8)	6.78%	(39)	3.57%	(13)	5.78%	(27)	4.21%	(26)
	CDR	0%	(0)	−33.22%	(39)	−56.84%	(13)	−21.39%	(10)	−71.52%	(11)
	GDR	0%	(0)	0.41%	(39)	−0.53%	(13)	−8.58%	(14)	−16.29%	(13)
KNeighbors	DVol	1.59%	(6)	−8.26%	(39)	−2.99%	(13)	−4.11%	(16)	−4.47%	(24)
	PU	−23.61%	(6)	−34.09%	(39)	−8.08%	(13)	−37.99%	(16)	−18.11%	(20)
	DTP	0%	(0)	−4.73%	(39)	−2.52%	(13)	−4.03%	(18)	−1.53%	(22)
	REst	−11.84%	(5)	−26.69%	(39)	−21.74%	(13)	−25.86%	(25)	−22.10%	(16)
	CDR	0%	(0)	9.44%	(39)	2.50%	(13)	7.35%	(11)	4.69%	(12)
	GDR	0%	(0)	−8.13%	(39)	−21.16%	(13)	−23.40%	(19)	−6.12%	(13)

Table 7. Effectiveness of different mitigation schemes measured in $\Delta\overline{NRMSE}$ and *#Retrains* (both are the lower the better) on CatBoost using both datasets. We show the best scheme of multi-group LEAF and denote it as LEAF*.

Dataset	KPIs	$\Delta\overline{NRMSE}$ (#Retrains) of Mitigation Schemes											
		ADWIN		Naïve_{30}		Naïve_{90}		Triggered		LEAF		LEAF*	
Fixed Dataset	DVol	−9.75%	(7)	−29.62%	(39)	−19.83%	(13)	−31.80%	(27)	−32.67%	(28)	−35.12%	(34)
	PU	−29.76%	(7)	−44.88%	(39)	−16.44%	(13)	−35.06%	(25)	−46.59%	(35)	−47.62%	(27)
	DTP	−1.16%	(4)	−20.02%	(39)	−16.88%	(13)	−23.84%	(28)	−24.30%	(31)	−24.64%	(30)
	REst	−16.01%	(8)	−35.41%	(39)	−26.25%	(13)	−38.38%	(25)	−38.44%	(31)	−41.27%	(32)
	CDR	0.00%	(0)	2.35%	(39)	−5.39%	(13)	−4.21%	(17)	−3.63%	(9)	−6.22%	(12)
	GDR	0.00%	(0)	3.37%	(39)	−4.20%	(13)	44.56%	(17)	−6.24%	(19)	−6.24%	(19)
Evolving Dataset	DVol	−8.15%	(6)	−29.00%	(39)	−19.53%	(13)	−30.76%	(24)	−32.09%	(37)	−32.80%	(30)
	PU	−28.26%	(6)	−44.56%	(39)	−17.38%	(13)	−50.89%	(24)	−45.75%	(24)	−51.72%	(26)
	DTP	−0.20%	(4)	−19.51%	(39)	−17.59%	(13)	−22.19%	(30)	−22.58%	(27)	−22.58%	(27)
	REst	−17.00%	(7)	−37.51%	(39)	−28.33%	(13)	−44.01%	(25)	−43.66%	(26)	−48.01%	(33)
	CDR	−2.70%	(4)	−3.79%	(39)	−1.24%	(13)	−6.79%	(7)	−1.33%	(9)	−7.15%	(8)
	GDR	2.15%	(4)	−8.46%	(39)	−2.65%	(13)	−13.21%	(15)	−2.06%	(13)	−11.99%	(17)