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[§]
University of Chicago, [†]École Normale Supérieure de Lyon,
[‡]University of Hawaii, [§]Verizon
{shinanliu, abhagoji, feamster}@uchicago.edu, francesco.bronzino@ens-lyon.fr, pschmitt@hawaii.edu,
{hector.garcia, timothy.coyle, brian.ward2}@verizon.com

ABSTRACT

Operational networks commonly rely on machine learning models for many tasks, including detecting anomalies, inferring application performance, and forecasting demand. Yet, unfortunately, model accuracy can degrade due to *concept drift*, whereby the relationship between the features and the target prediction changes due to reasons ranging from software upgrades to seasonality to changes in user behavior. Mitigating concept drift is thus an essential part of operationalizing machine learning models, and yet despite its importance, concept drift has not been extensively explored in the context of networking—or regression models in general. Thus, it is not well-understood how to detect or mitigate it for many common network management tasks that currently rely on machine learning models. Unfortunately, as we show, concept drift cannot be sufficiently mitigated by frequently retraining models using newly available data, and doing so can even degrade model accuracy further. In this paper, we 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. To do so, we develop Local Error Approximation of Features (LEAF). We introduce LEAF and demonstrate its effectiveness on a variety of KPIs and models. LEAF detects drift; explains features and time intervals that most contribute to drift; and mitigates drift 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 initial tests with a major cellular provider in the US show that LEAF is effective on complex, real-world data. LEAF consistently outperforms periodic and triggered retraining while reducing costly retraining operations.

1 INTRODUCTION

Network operators rely on machine learning models to perform many tasks, including anomaly detection [49, 50], performance inference [22] and diagnosis, and forecasting [14, 40]. Deploying and maintaining the models can prove challenging in practice [52]; a significant operational challenge is *concept drift*, whereby a model that is initially accurate at a particular point in time becomes less accurate over time—either due to a sudden change, periodic shifts, or gradual drift. Previous work in applying machine learning models to network management tasks has generally trained and evaluated models on fixed, offline datasets [5, 14, 17, 40, 49–51], demonstrating the ability to predict various network properties or instances 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.

Models can become less accurate at predicting target variables for many reasons. One cause is a sudden, drastic change to the environment. For example, the installation of new equipment, a software upgrade, or a sudden change in traffic patterns or demands can cause models to suddenly become inaccurate. One notable instance that exhibited such a sudden shift was the COVID-19 pandemic, an exogenous shock that resulted in significant changes to behavior patterns [38] (*e.g.*, reduced mobility) and traffic demands [34] (*e.g.*, as people suddenly began using home WiFi connections more for specific applications and relying less on mobile data). Another possible cause is periodic change, whereby a model that is accurate on a particular time window may become less accurate but will exhibit higher accuracy at fixed, periodic intervals in the future. For example, one clear phenomenon that we observe is drift in model accuracy with a seven-day period; diurnal and periodic patterns in network traffic are, of course, both well-documented and relatively well-understood. A new contribution in this paper, however, is to demonstrate that machine learning models also exhibit similar patterns with respect to model accuracy.

Concept drift is a relatively well-understood phenomenon in machine learning and has been studied in the context of many other prediction problems [24, 36, 48]. Yet, although concept drift is a relatively well-understood phenomenon in other fields, it has not been explored in the context of computer networking, in spite of the relatively widespread use of machine learning models.

Mitigating concept drift for Key Performance Indicators (KPIs) in a cellular network introduces fundamentally new challenges that make previous approaches from other domains inapplicable. In contrast to tasks such as image or text classification, where the semantics of prediction occurs on a fixed object and characteristics of the features change relatively slowly over time [19, 24, 60], predictions of network characteristics occur continuously, and occur within the context of a system that changes over time due to both gradual evolution and exogenous shocks. Networks have unique characteristics, such as dynamic signal interference due to environment changes (*e.g.*, changes due to weather, seasonality, etc.) [59], which calls for new approaches. One important implication of our findings is that one of the most common practices for combatting drift in large networks—regularly retraining models [30, 53]—is often not sufficient and can in some cases even reduce model accuracy.

Beyond simply detecting and mitigating concept drift, operators often want to know *why* a model has become less accurate. Thus, the ability to explain the behavior of black-box regression models is necessary to help operators use these models in practice. To this end, this paper not only characterizes concept drift in cellular networks, but also develops (1) approaches to explain how different features contribute to drift; and (2) strategies to mitigate drift through triggered, focused re-retraining, forgetting, and over-sampling approaches that are more efficient than naïve retraining. Towards this goal of *explainable AI (XAI)* models in networking, this paper studies drift in the context of cellular networks and develops new techniques to mitigate concept drift. Although past work has developed methods to help explain concept drift, it has largely focused on classification problems [29, 58]; in contrast, prediction problems in the context of cellular networks are often regression problems, which generally lack methods to explain concept drift.

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. Further, the diverse, longitudinal nature of the dataset used presents a challenging concept drift problem.

Second, **we introduce *LEAF* (Local Error Approximation of Features) to detect, explain, and mitigate concept drift in cellular networks**. We apply Kolmogorov-Smirnov Windowing (KSWIN) to time-series of estimated errors, which tells us when a model drifts. We complement existing drift explanation methods in regression-based supervised learning tasks. 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 undertrained region of a model. Based on these explanations, the framework strategically re-samples the training data, and creates temporal ensembles to mitigate drift.

Third, **we evaluate LEAF holistically, on more than four years of data from a large cellular carrier, evaluating both its effectiveness and efficiency**, across many KPIs and families of machine learning models. LEAF consistently outperforms existing mitigation approaches, while reducing costly retraining operations by as much as 76.9% (compared to periodic retraining). Using explanations provided by groups of multiple representative features, we consistently reduce errors across KPIs for boosting, bagging, and LSTM-based models. We also find that while KPIs with higher dispersion are harder to mitigate, LEAF is still able to consistently reduce errors.

Our initial tests with a major cellular provider in the US show that LEAF is effective in practice. We believe that LEAF can also be 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 present in our dataset that LEAF tackles successfully. Such avenues present rich opportunities for future work.

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 datasets 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 these variables 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 the LTE architecture) 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 Fixed Dataset with a growing number of eNodeBs) in the remainder of the paper.

Collection period	Jan. 1st 2018 – March 28th 2022
Identifiers	eNodeB ID & Time stamp
Number of KPIs	224
Groups of KPIs	Resource utilization Network performance User experience
Number of eNBs	Fixed Dataset: 412 common eNBs Evolving Dataset: 898 eNBs
Number of logs	Fixed Dataset: 699,381 Evolving Dataset: 1,084,837

Table 1: *Summary of datasets.*

Why these datasets? The first requirement for any dataset used to analyze concept drift is that it should actually contain clear evidence of such drift. Our analysis in Section 3 establishes that the variables, which are Key Performance Indicators (KPIs) from the cellular network, in Fixed Dataset and Evolving Dataset do indeed drift over time. Intuitively, this drift occurs due to endogenous reasons like changes in network infrastructure and KPI definitions, as well as exogenous ones such as the COVID-19 pandemic.

Further, Fixed Dataset is affected by factors from software upgrades to mobility pattern changes. It is extremely diverse since different KPIs exhibit drastically different behavior over time. Some are relatively stable throughout, others change gradually while some are bursty, with sudden, short-lived changes in value. 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. Evolving Dataset extends the KPI diversity further by including the operational growth of eNodeBs in this area, adding more heterogeneity from practice.

Further details. 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. Evolving Dataset has the same information, but with a maximum of 898 eNodeBs, containing 1,084,837 daily logs.

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, VoLTE user satisfaction rate). Further, some of the KPIs have separate directional measurements, such as the downlink where the network is transmitting the data down to the UEs and the uplink where the network is receiving data from the UEs. The data also contains features like unique identifiers of dates, eNodeBs, and their characteristics (*e.g.*, the density of their deployment location).

Data pre-processing. We transform the timestamps to not only date, but also day of the week, month, and year to the feature sets, to help learners understand the temporal context of the data. Then, we discard string-based features (*e.g.*, eNodeB ID) because they might generate undesired bias that could affect accuracy when predicting eNodeBs not present in the training set, *i.e.*, text representation might be encoded to representations which are not meaningful for base stations that are not present at all times in the dataset.

2.2 Modeling Goal

Forecasting Problem. We focus our study of concept drift in the context of network forecasting. Network forecasting (capacity, performance, user experience) is an important problem for operators as it guides infrastructure configuration, management, and augmentation. We focus on per-eNodeB level KPI forecasting to provide suggestions for capacity adjustment, deployment, maintenance, and operation in large cellular networks.

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 use a 180-day forecast window because the model outputs we focus on are used *in practice* for network infrastructure provisioning and augmentation over long timescales. For such tasks, operators need at least 180 days to plan, decide, prepare, and execute capacity augmentation.

The nature of this problem is regression with time-series information. Regression models are a better fit than classification because (1) we aim to forecast target KPIs that are all characterized wide ranges of possible numerical values and (2) fine-grained forecasting better enables operators to understand the KPIs and take action on their network.

Forecasting Targets. We choose the forecasting targets based on their measurement goals. We focus on the prediction of six KPIs out of the available 224, two out of each of the three different KPI groups that are most relevant for

Property	Resource Utiliaztion		Network Performance		User Experience	
	DVol	PU	DTP	REst	CDR	GDR
Std/Mean	0.73	1.34	0.57	0.77	1.35	2.12
Periodic	✓	✓	✓	✓	✓	✓
Bursty		✓			✓	✓
Data Lost		✓				
Balanced	✓	✓	✓			

(a) *Fixed Dataset*

Property	Resource Utiliaztion		Network Performance		User Experience	
	DVol	PU	DTP	REst	CDR	GDR
Std/Mean	0.81	1.76	0.59	0.85	2.48	8.52
Periodic	✓	✓	✓	✓	✓	✓
Bursty		✓			✓	✓
Data Lost		✓				
Balanced		✓		✓		

(b) *Evolving Dataset*

Table 2: *Characteristics of target KPIs in both datasets. 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.*

use in 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 2 summarizes the main characteristics of these KPIs in both Fixed Dataset and Evolving Dataset. These KPIs exhibit a variety of statistical patterns and characteristics which, as we will see in Section 3 and 5 can ultimately affect how models drift over time, as well as the best strategies for mitigating drift for a particular KPI. For Fixed Dataset, 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. Some data for PU was lost between July 2019 and January 2020. For comparison, Evolving Dataset has noticeably higher dispersions (Std/Mean) on PU, CDR, and GDR, due to increased infrastructure in this dataset. Moreover, the distributions of DVol and DTP in it are more skewed than in Fixed Dataset.

Model Selection. To explore the performance of different widely-adopted regression techniques, we use for all experiments in this paper the AutoGluon [1] pipeline and TensorFlow. AutoGluon is used to quickly prototype deep learning and machine learning algorithms on various existing frameworks: LightGBM, XGBoost, CatBoost, Scikit-learn, MXNet, and FastAI. For all selected models, it enables automatic data processing, architecture search, hyperparameter tuning, and model selection and ensembling. 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 like 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 viable 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. For example, the LSTM network is implemented in TensorFlow, which has 100 units for the LSTM, followed by a dropout layer and a dense layer.

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 present in our dataset up to the date when the model is generated. While we target the forecasting of the target KPIs on an eNodeB-by-eNodeB basis, we generate a single model for the entire network, *i.e.*, we create a single model capable of forecasting values for each individual base station.

2.3 Metrics for Evaluating Drift

The characterization of model performance over time is essential to evaluate concept drift. Given that we model network capacity 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 and derive the Normalized Root Mean Squared Error (NRMSE):

$$NRMSE(M,t) = \frac{\sqrt{\frac{1}{N_t}\Sigma_{j=1}^{N_t}(M(k_{tj}) - \widehat{M(k_{tj})})^2}}{\max_j(k_{tj}) - \min_j(k_{tj})} \tag{1}$$

Here we denote M as the model we aim to evaluate, t denotes a specific date, and k refers to the target KPI being forecasted, k_t is the subset on date t, N_t is the number of data points on this subset, $M(k_{tj})$ denotes the jth measurement, while $\widehat{M(k_{tj})}$ identifies the jth prediction of M.

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.*, over-provision of resources); 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 [42]. To avoid possible errors in interpretation, we also test the performance of regression by the other metrics including coefficient of determination (*i.e.*, R^2), 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.

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:

$$\Delta\overline{NRMSE}(M_1, M_0) = \frac{\overline{NRMSE}(M_1) - \overline{NRMSE}(M_0)}{\overline{NRMSE}(M_0)} \times 100\% \tag{2}$$

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. We then study how different training set sizes and training periods can affect model performance. 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 mitigating concept drift. 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.

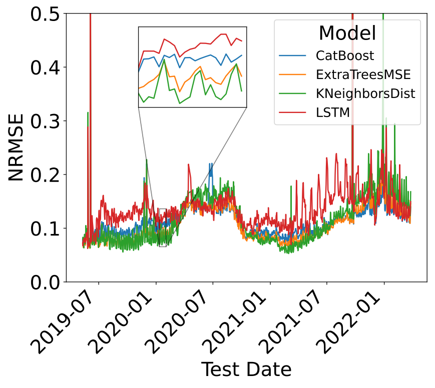
3.1 Comparing Drift Behavior across KPIs

We explore whether we can observe drift for the KPI forecasting task and additionally whether the drift patterns from different KPIs are similar. We experiment with different KPIs. We train all the models mentioned in Section 2.2 for each target KPI. To keep all other variables the same, the inputs of models are completely unchanged, but we alter the target KPI to predict. 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 on each date might be different, because of natural expansions of infrastructure in cellular networks.

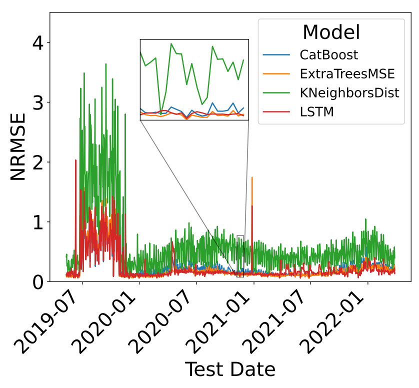
Uncorrelated KPIs exhibit different drift patterns. Figure 1 presents the concept drift along with time for three categories of KPIs. Overall, the drift patterns are quite unique for each class, and they vary in two aspects. First, deviations in NRMSE occur at different periods of time. In Figure 1a, NRMSE of downlink volume experiences a sudden shift in April 2020 (corresponding to the COVID-19 quarantine period, which is a unique example of sudden drift), and gradually drifts back to normal values in later months of 2020. Starting from March 2021, the NRMSE gradually increases and peaks around January 2022. For the prediction of peak users, Figure 1b demonstrates that July 2019 to November 2019 are harder to predict, because of lost data. The throughput prediction model in Figure 1c witnesses a rather stable NRMSE series. Figure 1e and Figure 1f exhibit higher values and larger fluctuations first, but the model stabilizes and shows improved NRMSE after April 2020. Moreover, short-lived, abrupt increases in error are more frequent than other KPIs, due to the burstiness of CDR and GDR. Unlike CDR, GDR has higher errors before COVID-19 lockdowns and lower errors afterwards.

Second, the high-frequency components have different patterns for different KPIs. By using signal processing techniques like STFT, no obvious weekly pattern is found on NRMSE of CDR and GDR. But if we look at the 3-week insets of the other KPIs, three repetitions of similar signal patterns appear, which indicates a weekly pattern.

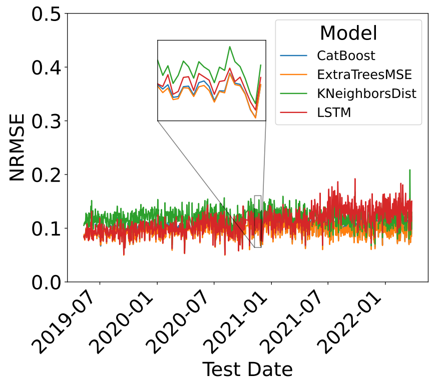
Different KPIs are most effectively predicted using different models. As shown in Figure 1, we can find a model that performs relatively well for each target KPI. For all the



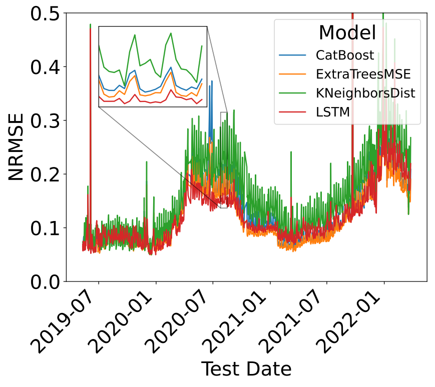
(a) *Volume.*



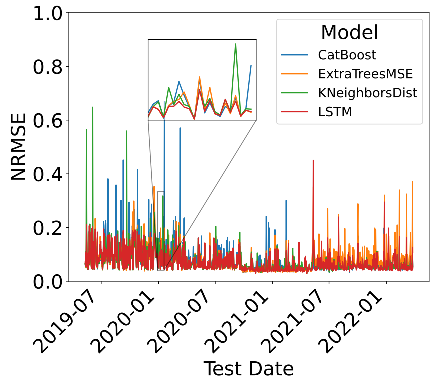
(b) *Peak active UEs.*



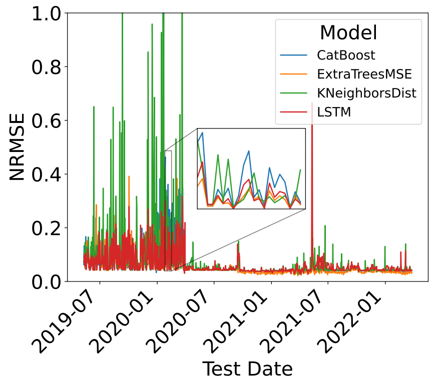
(c) *Throughput.*



(d) *RRC Establishment.*



(e) *Call drop rate.*



(f) *Gap duration ratio.*

Figure 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 accomodate larger errors in Fig.1b, 1e, 1f.*

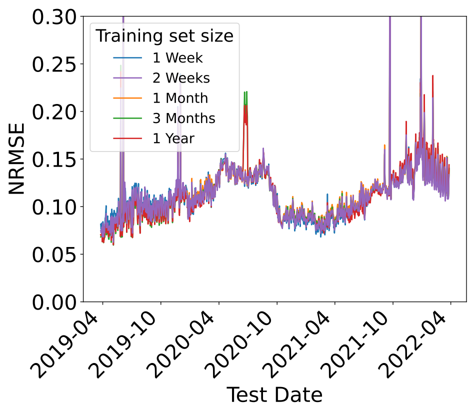
target KPIs, there is at least one model with NRMSE less than 0.1 (indicating good prediction power [42]) for at least one year. For example, even for GDR, the KPI that is most challenging to accurately predict (due to high coefficient of variances / dispersion), the average NRMSE from ExtraTrees is 0.055. And CatBoost in volume prediction has an average NRMSE of 0.116, with below 0.1 on 473 days.

3.2 Analyzing Drift for a Single KPI

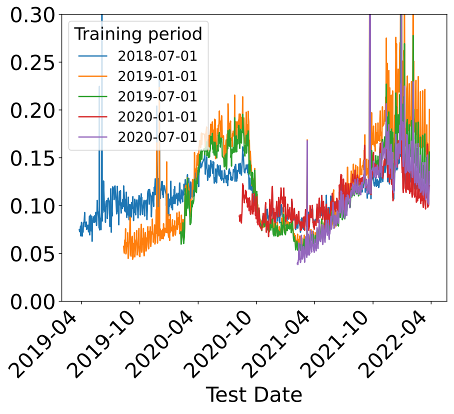
In this section, we analyze how changes in the model used, and how it is trained, impact concept drift. We use a single KPI (downlink volume) as an illustrative example.

Individual KPI predictions drift consistently across different models. As mentioned earlier, 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. 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 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



(a) *Training set size.*



(b) *Training set period (training set size = 2 weeks).*

Figure 2: *Effects of training set size and training set period on concept drift for the downlink volume KPI*

lockdown and gradually recovers after October 2020, then increases again from March 2021 and peaks at January 2022.

In Figure 2a, we see that the model effectiveness 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 optimizes the tradeoff between performance over time (*i.e.*, robustness) and efficiency.

Consistent drift appears when trained on different periods. Since we are testing the long-term effectiveness of models, another concern is whether the choice of the training period would affect the drift. In this experiment, we train on different 14-day windows of historical data from all eNodeBs. In Figure 2b, the legend shows which 14 days are selected for training: 14 days of data before that date. For each training set, the model is tested daily using a 180-day forecast.

Figure 2b suggests that, no matter which training period, all NRMSE values drift simultaneously at similar timestamps. Our experiment also suggests that *models trained on more recent time periods do not necessarily result in better model performance*. For example, after COVID lockdown, a more recently trained model (green line in Figure 2b) performs worse than an earlier model (blue line). During training, the green line uses all the data from June 17th to July 1st, 2019 as features and the downlink volume from December 18th to January 1st, 2020 as the forecasting target. This means that the model is trained on a rather abnormal period—the winter holiday before the beginning of the COVID-19 lockdowns. Doing so results in less accurate model predictions on the following test dates, especially when drastic changes brought by the COVID-19 quarantine begin. On the contrary, the model trained on older data (blue line in Figure 2b) preserves the performance better during COVID-19 lockdowns. This observation opens new possibilities (*e.g.*, optimal sample selection) for future research in exploring the best strategies for adapting to drift.

3.3 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 [30, 53] adopt this approach. To understand the effectiveness of this approach for the longitudinal cellular dataset we are working with, we retrain a number of different models using different retraining frequencies. Our aim is to understand how frequent retraining has to be in order to be effective, and if simply increasing the frequency is enough. Note that we do not fine-tune the models because fine-tuning is very costly for retraining and thus impractical, due to the need for manual adjustments from human experts. For this experiment, we use a training set of 14 days of data 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 day.

Somewhat counter to conventional practice, we find that simply retraining the model at regular 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 3, 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, frequent retrains are

Retraining	$\Delta\overline{NRMSE}$ of Target KPIs						#Retrains
Period	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

Table 3: *Changes of average NRMSE and number of retrains, over time, for different periodic retraining strategies.*

very costly for the network operator. Further, we observe that this trend breaks 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 just naïvely uses all of the most recent data can even adversely affect model performances.

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 necessarily result in better performance (Section 3.2). Further, complete data replacement neglects finer-grained error information across samples, throwing away useful samples from the past.

Overall, we conclude that while 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 fine-tune 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*, a framework for drift detection, explanation, and mitigation. We seek to leverage explainable AI methods to provide us with insights about drift in order to mitigate it effectively. We describe the core intuition behind LEAF, followed by the technical details for each component. As a walk-through example, we use a concrete case (applying the CatBoost model to downlink volume forecasting) to illustrate the framework in an operational setting.

4.1 Insights Behind LEAF's Design

LEAF aims to work with any regression-based supervised model, and provide explanations for black-box models. Based on the explanations provided, targeted mitigation strategies can be applied. While concept drift is not a new problem, existing solutions face two major limitations: (1) previous concept drift explanations are limited to classification problems; (2) concept drift mitigation is often coarse-grained, only focused on the global performance metrics.

Design choice 1: drift explanation for black-box regression models. Drift explanation is a relatively new field. In the context of malware detection, recent research has developed techniques to explain concept drift. Transcend [29] and CADE [58] use decision boundary-based or distance-based explanations to discover the feature sets that contribute most to drift. Unfortunately, both explanation approaches only apply to classification problems, and do not apply to prediction in general, such as the regression-based problems we study in this paper. Moreover, these methods only focus on explanations, instead of leveraging drift explanation to perform targeted mitigation.

To extend the scope of the explanation, we envision an approach to characterize errors of a regression model simply based on the model input and output. The common challenge in explaining the performance of machine learning-based solutions is that only a subset of algorithms are interpretable [41]. Unfortunately, in practice, a wide variety of uninterpretable models are used (*i.e.*, outsourced models such as GPT-3 [11], or black-box models). Model-agnostic methods (such as Partial Dependence Plot (PDP) [21] and Accumulated Local Effects (ALE) plot [3, 4]) overcome this issue by providing visual clues to the effect of different features and set of features on the predictions of black-box models. However, neither of these methods are able to look into local errors (distance of predictions and ground truths) instead of just predictions.

Design choice 2: mitigation based on local error decomposition. Previous techniques consider global information (*e.g.*, error metrics like RMSE, R^2; distribution-level

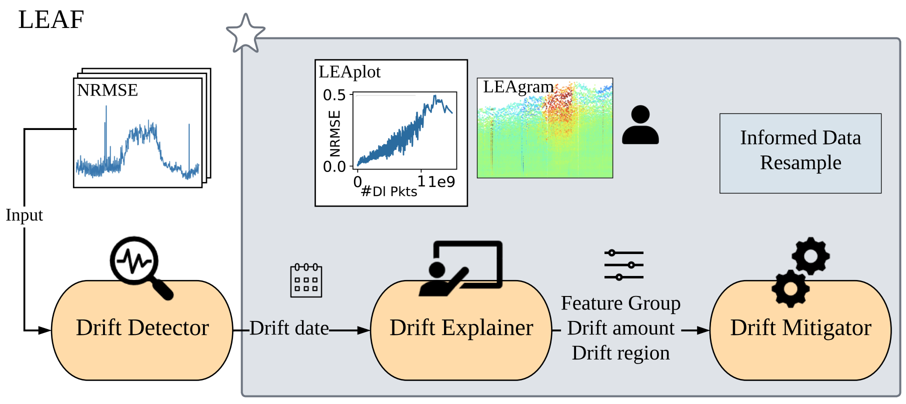


Figure 3: *The LEAF framework that detects (Appendix A), explains (§ 4.3), and mitigates (§ 4.4) concept drift. Our main contributions are in the gray box marked with a star.*

metrics like KL-divergence, etc.) as a signal of drift and use it in a coarse-grained manner. For example, Paired Learners [6] use one stable learner to learn on old data and another reactive learner to learn from the latest data, which has no fine-grained selection of weak predictions within samples. Our insight is that while the global error metrics provide a good measure of the performance of a black-box model over time, *the distribution of local errors across samples on each given time instance may be uneven.* Local errors may vary in different value ranges of a given feature or correlated feature set. Our goal is to determine the extent of error present in specific regions of the feature space, so we can deal with these error regions strategically (*e.g.*, forgetting, over-sampling).

Therefore, LEAF is proposed based on a simple idea: when concept drift is detected, the amount of *local error may vary* on the feature space and thus needs to be mitigated in a *targeted* manner based on statistical patterns. In order to achieve this, we (1) detect the shift in the time series of performance metrics such as NRMSE; (2) find the most representative features that affect drift; (3) determine the error distribution of these features; (4) understand the temporal changes of error distributions; and (5) make informed mitigation (*e.g.*, forgetting, over-sampling) decisions based on the weakly performed feature spaces. We describe LEAF in more detail in the following sections.

4.2 LEAF Framework Overview

LEAF compensates for concept drift in machine learning models by implementing a pipeline of three sequential components: (1) a drift *detector*; (2) a set of tools to *explain* drift for features; and (3) a drift *mitigator*. Note that LEAF does not require the use of any specific model nor internal access to the employed model. Instead, it solely requires 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 in-use model in the form of NRMSE time-series to determine whether drift is occurring. The detector applies the well-known Kolmogorov-Smirnov Windowing (KSWIN)[1] method [45, 54] on the time-series to identify a change in the distribution of the output error, providing an indicator of whether drift is occurring. Although drift detection is critical, it is not the main focus of this work, as there is significant prior work on drift detection. A detailed discussion of drift detection is deferred to Appendix A as the main research contributions of this work are in the areas of drift explanation and informed mitigation. Once drift is detected, the explainer is triggered, indicating the time instance at which drift occurred. The explainer determines the most representative features that contribute to drift, and then uses Local Error Approximation (LEA) to characterize the drift, and offer guidance for model compensation. It also contains visual tools (*i.e.*, LEAplot and LEAgram) to provide insights on the time intervals where drift is most severe. Based on the error distribution and statistical patterns, the mitigator *automatically* forgets previous data, and performs informed replacement by sampling/over-sampling targeted regions.

We design LEAF to use discrete components to enable network operators to use any or all of LEAF's components to gain insights in their existing prediction pipelines. They can be notified of the detection of drift. They can take the output of the drift explainer to further understand drifting

[1]We apply the most effective drift detection techniques in this well-explored area [8, 9, 26, 36]. We also tested ADWIN, DDM, HDDM, EDDM, PageHinkley, but KSWIN was the most effective on our NRMSE series.

features, their severity, and the regions in the dataset experiencing the most drift. Finally, they can use the mitigator to automatically obtain metrics towards informed mitigation. After a monitoring phase, operators can either decide on an appropriate manual course of action based on the metrics that LEAF provides, or allow LEAF to automatically replace the original model.

4.3 Explaining Drift

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 operational costs and decompose them in a spatio-temporal manner. The LEAF explanation module is based on the idea of local error decomposition: in any regression-based black-box model, local errors may occur in a specific range of a given feature or correlated feature set. Once drift is discovered in the NRMSE time-series, LEAF determines the extent of error present in specific regions of the feature space.

Multi-group correlated features. Natural correlations of features are often part of a dataset with a large number of features [27, 55]. 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 are able to obtain a set of representative features.

To determine if the drift explaining features carry the correct semantic meanings, we manually examine with operators in a case study. We apply CatBoost to predict downlink volume, same as the rest of this section. Table 4 shows the top three correlated feature groups after the above grouping process. The most representative feature of the 1st group is `pdcp_dl_datavol_mb`, the history of downlink volume itself. Highly correlated features include different stages of traffic communication: side channels that carry or reflect volume (*e.g.*, average downlink packets from UE `avg_ue_downlink_pkts`), connection establishments and releases (*e.g.*, establishments of RRC `rrc_establishmentsucc`), and etc. All 32 features from this group contribute to error jointly, and thus will be used to explain the drift as a whole. Representative features from other groups provide different views of possible error reasons. The second group contains features that reflect another reason: `badcoveragemeansurements`, which measures the bad coverage of eNodeBs. This often raises 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, hence downlink volume.

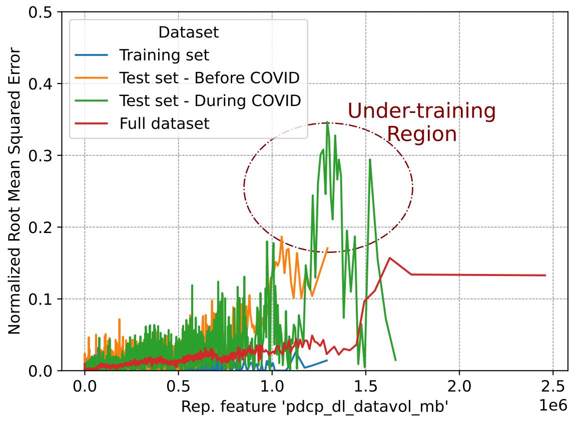
Local error approximation (LEA). To decompose the error of a model on any corresponding dataset, we use the selected representative features as features to inspect. For each feature, we group samples based on the value of the representative feature into N bins (*i.e.*, quantiles). The higher N is, the 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 aggregate these N measurements into a vector to represent the error distribution of each local quantile. This technique yields an approximation of local errors over the range of the most sensitive and representative feature(s) for a certain model and corresponding dataset. LEA characterizes the extent of drift and opens up opportunities to mitigate it in a targeted fashion. LEA, together with the feature grouping, provides the foundation of the LEAF framework.

LEAplot and LEAgram. Starting from LEA, we develop LEAplot and LEAgram to assist LEAF users to visualize local error components, compare them across different data subsets, and, overall, better understand root which features most impact the occurrence of model drift. Figure 4 shows the LEAplots of representative features from the top two feature groups of our case study. In Figure 4a, `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 below 1e6, the errors in "During COVID" test set are more than 10x those of the training set and nearly 1.5x of those in the "Before COVID" test set above 1.2e6. Figure 4b, however, shows the LEAplots of representative feature `badcoveragemeasurements` from Group 2. When the value is above 2e5, "During COVID" 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.

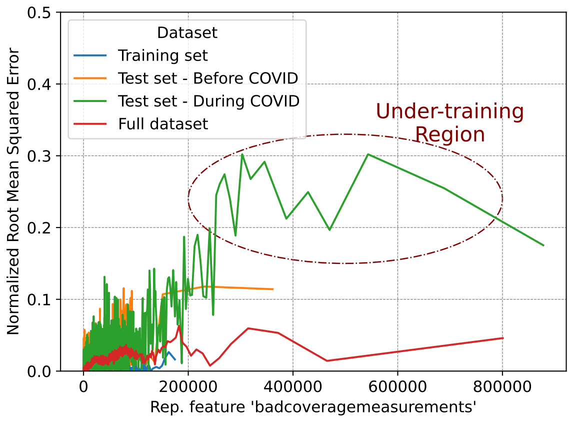
These two LEAplots exhibit very different error distributions. For example, samples from the high error region of Figure 4a are not always in that of Figure 4b. It indicates that we can leverage this difference for iterative mitigation on different feature groups. On the contrary, the features

Feature group	Features (rank by importance)
1	**pdcp_dl_datavol_mb**, avg_ue_downlink_pkts, avg_ue_uplink_pkts, rrc_establishmentsucc, pdcp_vol_ul_drb_mb, s1_establishmentatt, activesessiontime_minutes, bearer_setup_failure_%_qci5_den, initial_erab_establishmentsucc, erab_rels_normal_enb, $\cdots$
2	**badcoveragemeansurements**, $\cdots$
3	**rtp_gap_ratio_medium%**, $\cdots$

Table 4: *Top 3 correlated feature groups of CatBoost model that predicts DVol. The most representative features of each group are marked in bold.*



(a) *Feature group 1.*



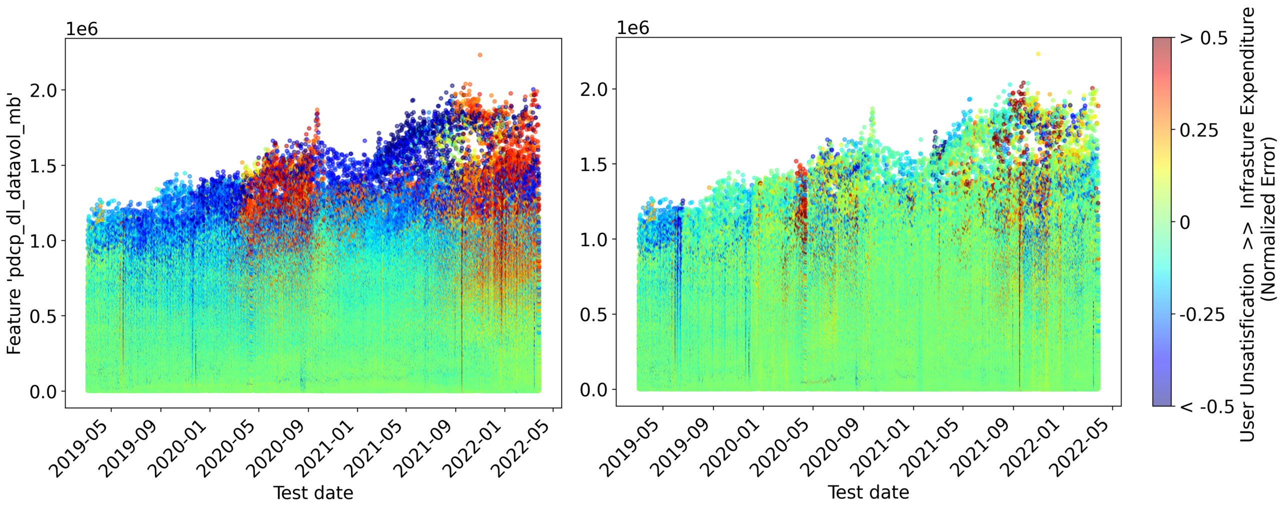
(b) *Feature group 2.*

Figure 4: *The LEAplots (1,000 bins) that decomposes CatBoost NRMSE time-series in Figure 2a. 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).*

from the same group have similar error distributions (Figure 7 in Appendix B), which validates the feature grouping process mentioned above: naturally correlated features contributed to the error together, and thus should be considered for mitigation as a whole.

LEAgram augments LEAplot by incorporating temporal information along with error information. It shows the error for individual samples in the entire test set, arranged temporally. The test set is divided by time interval (*e.g.*, date) and assigns samples from those divided datasets into N bins, based on the quantiles of the most important feature with regard to drift. NRMSE is computed within each bin. When N is greater than or equal to the number of samples on each time interval, NRMSE is equivalent to Normalized Mean Absolute Error (NMAE).

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 improve the NRMSE-based approach to create LEAgrams with two modifications: first, we expand the choice of bin N to be always greater than the number of samples to see individual sample effects; second, we change the metric to Normalized Error (NE) to preserve the sign. Figure 5a shows the LEAgram which explains the performance of CatBoost. 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. 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 that infrastructure will face lower demand than in reality.



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

Figure 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 decompostion of CatBoost NRMSE. (b) shows errors of mitigated CatBoost model if LEAF applied.*

4.4 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.*, coefficient of variance or Std/Mean in Table 2) of the target KPIs.

Forgetting and over-sampling. 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). For example, errors (NRMSE) are computed for N bins of the feature range, and they are associated with each bin. This error distribution E_L shows the area of the latest dataset that is inaccurately predicted or under-training. We first develop a strategy to forget. Because it is a continuous process, we apply this method to the last training set. We assign weights to each sample based on the bin separations of E_L that it falls into. So that the weight distribution E_p is proportional to the area of the error. In those areas with high error contributions, we throw away the samples with low weights. Then we introduce over-sampling into the pipeline from the existing collected dataset (including the latest drifting samples) based on weights provided by E_L. Afterwards, retrains are 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 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 needs to be more aggressive. For example, for the target KPIs exhibiting the coefficient of variation higher than 1 (*e.g.*, PU, CDR, GDR) in the latest dataset, we forget the samples of the original dataset with linear weights assigned in E_p. In terms of over-sampling, we explicitly use non-linear (cubic) weights [33] of E_L to focus on the regions of high error in the latest drifting instances. However, if the dispersion of a target KPI 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 E_L to over-sample the latest drifting instances. Note that these thresholds and sample fractions are optimal to our dataset and might need to be tuned for other datasets.

End-to-end Example Mitigation Case Study. Given our downlink volume use case, we demonstrate the mitigation performance of LEAF. Figure 5b shows the mitigating effects of LEAF on CatBoost. Once the mitigation is triggered successfully at the beginning of December 2019, and also after October 2020, the under-estimation above 1e6 is eased through strategic forgetting and over-sampling. Between March 15th, 2020 and October 1st, 2020, and after October 2021 the extreme over-predictions above 0.75e6 are largely reduced. Overall, Figure 5b shows a 32.68% of reduction in $\Delta\overline{NRMSE}$ compared to Figure 5a, with a major mitigation focus of the errors at the tail. This demonstrates the effectiveness of this end-to-end mitigation strategy.

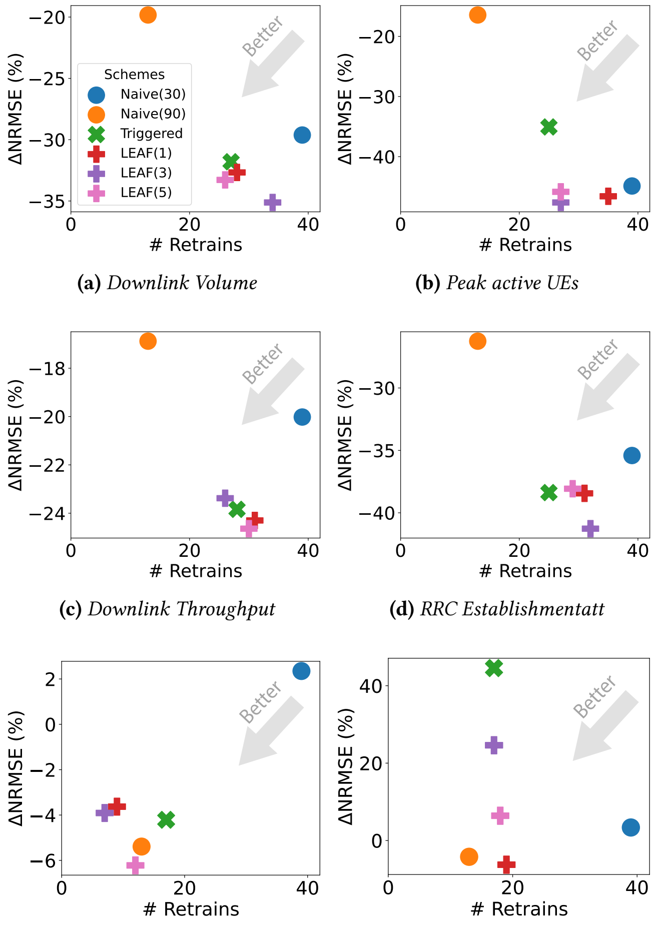
5 EVALUATION

In this section, we evaluate how the LEAF framework mitigates drift compared to baseline techniques. We present how different mitigation schemes improve end-to-end model performance over the Fixed Dataset. 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. We focus on the comparison across mitigation schemes as Section 3 already established the need for applying techniques to combat drift. A comparison between LEAF and the static model is available in Appendix C.

5.1 End-to-End Comparison Across Mitigation Schemes

In this subsection, we compare LEAF's performance against different practical mitigation schemes, including the state-of-the-art approach presented in Section 3.3, *i.e.*, naïve retraining. We evaluate how mitigation schemes behave across both effectiveness in sustaining model performance over time as well as their retraining cost, *i.e.*, how many times they require a full model retrain. We present results for four models (one per model family) and the six target KPIs (Table 2).

We compare the average NRMSE over the duration of the dataset against two baselines: the naïve retraining scheme (see Section 3.3) that retrains the model every 30 or 90 days (we choose these two frequencies because they present the best performance versus retraining frequency); and triggered retraining, which only utilizes the drift detector information, *i.e.*, retrain the model using the latest available data whenever drift is detected. For LEAF, we also control the number of feature groups used during the mitigation phase. We show results for one, three, and five feature groups as they are the best performing configurations. To keep the evaluation fair across methods, we use the same amount of data for each retrain operation, which also controls the amount of time needed for a single retrain across schemes. The mitigation effectiveness is compared against a static model (trained using 14 days of data before July 1st, 2018) for each target KPI.



(a) *Downlink Volume*

(b) *Peak active UEs*

(c) *Downlink Throughput*

(d) *RRC Establishmentatt*

(e) *Call drop Rate*

(f) *Gap Duration Ratio*

Figure 6: *$\Delta\overline{NRMSE}$ vs. #Retrains under different mitigation schemes using CatBoost. For naïve retraining, number in parentheses denotes the retraining period in days. For LEAF, it represents the number of feature groups being used in the mitigation. Bottom left has the best mitigation effectiveness and lowest retraining cost.*

Figure 6 shows the trade-off between $\Delta\overline{NRMSE}$ and the number of retrains required by each scheme using CatBoost across KPIs. Such trade-off is of particular because it 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. Our goal is to find the scheme that achieves the best mitigation effectiveness first, while balancing the alternative goal of few retrains. As such, the bottom left of each subfigure represents the best option.

In the figure we observe that, as expected, naïve retraining every 30 days requires the highest number of retrains, while its mitigation effectiveness never outperforms LEAF. Naïve retraining every 90 days, however, requires the fewest retrains except for CDR mitigation. However, the mitigation

KPI.

effectiveness is frequently inferior to LEAF's, sitting in the top left corner in Figure 6a, 6b, 6c, and 6d. While this mitigation scheme seems to perform well for CDR and GDR, it does not achieve the best mitigation. The triggered mitigation scheme often lies in the middle, never outperforming other schemes across either metric. It also has exponential errors for KPIs like GDR, making it less practical among the schemes since it does not guarantee performance improvements for a model after mitigation.

Finally, LEAF consistently outperforms all baseline schemes. Thanks to its approach focused on using error explanation and informed mitigation, LEAF already exceeds the performance of other methods across KPIs, except for CDR, even when only using one representative feature. When considering additional feature groups, it can achieve either higher effectiveness or a lower number of retrains needed. Compared to triggered retraining, LEAF is able to 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. Notably, the number of feature groups used in LEAF also impacts mitigation effectiveness. This will be further discussed in the next subsection.

5.2 Sensitivity Analysis

In this section, we aim to understand which configurations work best depending on the number of feature groups employed in LEAF and its performance across different models and KPIs.

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 Figure 6, the amount of errors being mitigated tends 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. For DVol, PU, and REst, three feature groups lead to the highest mitigation effectiveness. For DTP and CDR, five feature groups work best. As the most dispersed KPI, GDR 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. Table 5 presents a summary of the $\Delta\overline{NRMSE}$ across different forecasting models (the lower, the better). We highlight in gray the best-performing scheme for each model and KPI. 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 models, 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 guarantee model 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. We validate this pattern by testing on other boosting (LightGBM) and bagging (Random Forest) algorithms and notice that it holds across these model families.

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 mitigation 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.

Finally, we observe that LEAF does not mitigate well KNeighbors (and other distance-based models). Naïve retraining every 30 days and triggered retraining using non-sampled fresh data perform better than LEAF across all KPIs using this model. We believe this behavior is rooted in the expressiveness of distance-based models and how they generalize [35]. KNeighbors employs a lazy regressor that memorizes all the training set and uses the least distance to the nearest neighbor to perform predictions. While over-sampling the error region can show targeted improvement, the originally good regions may perform worse because of the unbalanced added samples. In contrast, bagging methods like Extra Trees train weak learners in parallel [25] (or sequentially with boosting [18]). As individual learners learn new samples more independently, they are less affected by previous learners when targetedly mitigating error regions.

Different KPIs. As observed in the previous analysis, we find that different KPI time-series are mitigated more or less effectively depending on their own characteristics. For example, in CatBoost and ExtraTrees mitigation, PU has the highest NMRSE among KPIs across models and it is among the three most challenging KPIs to mitigate, along with CDR and GDR. This behavior can be intuitively attributed to the fact that PU has the highest NMRSE values over time caused

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

Table 5: *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.*

by sudden data losses. This generates an inherent high variability and causes model collapse. CDR and GDR have low baseline NMRSEs but are difficult to mitigate due to the high-frequency variation they experience. We validate this hypothesis by looking at the coefficient of variance, shown in Table 2, where PU, CDR, and GDR have the highest coefficients of variation of 1.34, 1.35 and 2.12, respectively. These KPIs, when mitigated by a method such as triggered retraining, actually lead to a large increase in NMRSE. Less dispersed KPIs like DVol, DTP, and REst, although still present drift, are easier to adapt, because of more homogenous distribution changes.

5.3 LEAF Effectiveness on Evolving Infrastructure

In this section, we present the effectiveness when applying LEAF on models that aim to forecast KPIs for an evolving infrastructure, *i.e.*, where new eNodeBs are being constantly deployed. 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.

We evaluate the effectiveness of the mitigation schemes across both datasets and present results in Table 6. As before, we only show the best scheme of multi-group LEAF and denote it as LEAF*. We observe that both naïve retraining schemes have very close performance across the majority of KPIs, with the exception of CDR and GDR with a retraining frequency of 30 days. We assume that this improvement can be recollected to the high retraining frequency of the first scheme that enables the model to quickly capture new eNodeBs as they are deployed. This trend is also observed for triggered retraining: working on the Evolving Dataset greatly improves performance across the majority of KPIs, confirming that employing a timely drift detector is beneficial to detect changes in the data when new elements are deployed

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

Table 6: *Effectiveness of different mitigation schemes measured in $\Delta\overline{NRMSE}$ and #Retrains (both are the lower the better) using both datasets. We show the best scheme of multi-group LEAF and denote it as LEAF**

in the infrastructure. Finally, we observe that both LEAF and LEAF* performance remains consistent across datasets, while also remaining the best performing mitigation strategy. This demonstrates that LEAF outperforms other baselines as it both integrates a detector to identify when drift is occurring as well as a more effective mitigation strategy that can use features error information to better target the data to use for retraining.

6 RELATED WORK

Concept drift is a pervasive phenomenon in machine learning that has been well-studied in various contexts and scenarios. We survey various aspects of concept drift, within networking and more broadly, as well as previous work in drift detection, explainable AI, and drift adaptation—all of which are increasingly active research areas.

Drift in network management. Machine learning has been applied to many networking problems, including anomaly detection [49, 50], intrusion detection [17, 51], cognitive network management [5], and network forecasting [14, 40]. Few studies have explicitly explored concept drift in the networking context. It is, however, well-known that network traffic is inherently variable over time. For example, mobile networks exhibit different patterns of activity based on time of day, day of week, and location in Rome, Italy [46]. Features such as latency can also drift over long periods of time [43]. 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, 20, 34, 38, 39]. Sommer and Paxson articulate that applying machine learning to anomaly detection is fundamentally difficult in large-scale operational networks [52] due to these (and other) challenges. All these factors contribute to concept drift in predictive models, yet, to our knowledge, this paper is the first to explore the effects of these types of changes on model accuracy for network management tasks.

Drift in other related fields. Concept drift has also been studied across other real-world contexts. For example, spam detection faces drift challenges, as spammers may actively change the underlying distribution of their messages [19]. Recommender systems are another context where drift occurs: user side-effects such as preferences and item side-effects such as popularity both change over time [24, 37]. Similar situations occur in monitoring systems [44], fraud [60], and malware detection [7, 29, 58].

Drift characterization and detection. Concept drift has been characterized both quantitatively based on probability distribution [24, 36, 56], as well as based on drift subject, frequency, transition, recurrence, and magnitude [56]. Sources of drift are also discussed using data distribution and decision boundary changes [36]. A long line of research has focused on detecting concept drift in deployed systems. A significant body of detection methods are based on error rate [36]. For example, Drift Detection Method (DDM) [23] provides a warning level and a detection level by detecting changes in online error rate within a landmark time window. Another approach is ADaptive WINdowing (ADWIN) [9], which uses two adjustable sliding windows to store older and recent data, and detect drift based on the differences between them. KSWIN [45] is a recent improvement upon ADWIN which uses KS-test to capture distances between windows. Large scale comparisons across drift detection methods have been conducted [8, 26, 45] showing the effectiveness of DDM and KSWIN. LEAF applies KSWIN to detect drift a large cellular network. LEAF's contribution is not to develop a new drift detection method; rather, the contribution of this work is in developing new explanation and mitigation techniques for cellular networks; integrating detection, explanation, and mitigation into a unified end-to-end framework; and evaluating the utility of various end-to-end mitigation strategies.

Explainable AI and drift explanation. Explainable AI has recently attempted to address the challenge of black-box model interpretation. Global model-agnostic methods, such as Partial Dependence Plot (PDP) [21] and Accumulated Local Effects (ALE) plot [3, 4], provide visual clues on the effect of different features and set of features on the prediction accuracy 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 (not just individual features), and (3) helping to visualize how drift evolves over time in an operational setting. Local agnostic methods, such as LIME [2, 47] and LEMNA [28], substitute a local model region with an interpretable one, enabling targeted patching to improve accuracy. Recent research has developed techniques to explain concept drift in the context of malware detection. Transcend [29] uses statistical comparison of samples to identify model decay thresholds (*i.e.*, decision boundary-based explanation). CADE [58] uses contrastive learning to develop a distance-based explanation to find the feature sets that have the largest distance changes towards the centroid in the latent space. Unfortunately, both explanation approaches only apply to classification problems, and do not apply to prediction in general, such as the regression-based prediction problems we study in this paper. To our knowledge, LEAF is the first framework to explain concept drift for regression-based prediction problems for network management tasks.

Drift mitigation. Adapting a model to migitate concept drift is also a well-explored area [24, 36]. Adaptation can be based on retraining or model ensemble. Retraining-based approaches often require complex data management and significant storage and memory. Online learning algorithms such as VFDT [16] keep the latest instance that adapts to slow changes. Paired Learners [6] use one stable learner to learn on old data and another reactive learner to learn from the latest data. DELM [57] adaptively adjusts the number of hidden layer nodes to combat drift during retraining. Ensemble methods have also been used for model adaptation [32]. Dynamic Weighted Majority [31] deals with non-stationary streams by allocating weights on different models. Accuracy Updated Ensemble (AUE2) [12, 13] incrementally updates sub-models on a small portion of data to adapt to drift. LEAF is inspired by several of these approaches, but focuses more heavily on identifying features that lead to concept drift, and performing mitigation at scale.

7 CONCLUSION

An important aspect of the applicability of machine learning models to networking tasks in practice is concept drift, which can occur periodically, suddenly, or gradually over time. Although this phenomenon has been explored in other contexts, it has received limited attention in the networking domain. To address this critical problem, this paper has developed, presented, and evaluated LEAF, a framework to detect, explain, and mitigate drift for machine learning models applied to networks. The LEAF framework employs explainable AI and informed mitigation; integrates them into an end-to-end drift mitigation system; and is evaluated for network forecasting problems in a large metropolitan area for one of the largest cellular networks in the United States. Our results based on more than four years of KPI data from this network show that LEAF consistently outperforms both periodic and triggered retraining while reducing the cost of retraining, across multiple types of regression models and KPIs. We believe that the LEAF framework can be applied beyond cellular networks, to other network management problems that use black-box models for regression-based prediction. Yet, these hypotheses are yet to be explored and thus are excellent avenues for future work. Another caveat is that the evaluation LEAF to date has been conducted on fully labeled datasets; thus, a promising direction could be to improve LEAF to cope with semi-supervised or unsupervised models with partial or no labels.

REFERENCES

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

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. 2020. Transcending transcend: Revisiting malware classification with conformal evaluation. *arXiv preprint arXiv:2010.03856* (2020).

[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] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[12] 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.

[13] 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.

[14] 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.*

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

[16] Pedro Domingos and Geoff Hulten. 2000. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining.* 71–80.

[17] 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.

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

[19] 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.

[20] 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).*

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

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

[23] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. 2004. Learning with drift detection. In *Brazilian symposium on artificial intelligence.* Springer, 286–295.

[24] 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.

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

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

[31] J Zico Kolter and Marcus A Maloof. 2007. Dynamic weighted majority: An ensemble method for drifting concepts. *The Journal of Machine Learning Research* 8 (2007), 2755–2790.

[32] 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.

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

[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] Jie Lu, Dianshuang Wu, Mingsong Mao, Wei Wang, and Guangquan Zhang. 2015. Recommender system application developments: a survey. *Decision Support Systems* 74 (2015), 12–32.

[38] 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.

[39] 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.

[40] 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.

[41] Christoph Molnar. 2020. *Interpretable machine learning*. Lulu. com.

[42] 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).

[43] 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.

[44] Mykola Pechenizkiy, Jorn Bakker, I Žliobaitė, Andriy Ivannikov, and Tommi Kärkkäinen. 2010. Online mass flow prediction in CFB boilers with explicit detection of sudden concept drift. *ACM SIGKDD Explorations Newsletter* 11, 2 (2010), 109–116.

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

[46] Jonathan Reades, Francesco Calabrese, and Carlo Ratti. 2009. Eigenplaces: analysing cities using the space–time structure of the mobile phone network. *Environment and Planning B: Planning and Design* 36, 5 (2009), 824–836.

[47] 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.

[48] Jeffrey C Schlimmer and Richard H Granger. 1986. Incremental learning from noisy data. *Machine learning* 1, 3 (1986), 317–354.

[49] 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.

[50] Taeshik Shon and Jongsub Moon. 2007. A hybrid machine learning approach to network anomaly detection. *Information Sciences* 177, 18 (2007), 3799–3821.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] Laura Toloşi and Thomas Lengauer. 2011. Classification with correlated features: unreliability of feature ranking and solutions. *Bioinformatics* 27, 14 (2011), 1986–1994.

[56] Geoffrey I Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. 2016. Characterizing concept drift. *Data Mining and Knowledge Discovery* 30, 4 (2016), 964–994.

[57] Shuliang Xu and Junhong Wang. 2017. Dynamic extreme learning machine for data stream classification. *Neurocomputing* 238 (2017), 433–449.

[58] 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)*.

[59] 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.

[60] Indrė Žliobaitė. 2010. Adaptive training set formation. (2010).

APPENDIX A 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) [45, 54]. KSWIN is a recent concept drift detection method [45] based on KS test, which is a non-parametric statistical test that makes no assumption of underlying data distributions [54]. It monitors the performance distributions as data arrives. We take the CatBoost NRMSE time-series to forecast downlink volume as an example; the model is trained on a total volume of 14 days of data before July 1, 2018 (illustrated in Figure 2a). Note that we do not have the ground truths of drifts in our dataset, thus we cross-check detection results with clear signals (*e.g.*, missing data, network changes due to COVID-19). By applying KSWIN, we observe that instances of drift are detected when the data exhibits major anomalies around June 2019, December 2019, and April 2021. The beginning and end of the COVID-19 quarantine period are also effectively detected. We tested KSWIN across the NRMSE time-series of the five KPIs of interest, using different model types and different training set sizes and periods. KSWIN performs well across all tasks.

APPENDIX B LEAF MULTIGROUP INFORMATION

Same group LEAplot. Natural correlations of features are often part of a dataset with a large number of features. These correlated features contribute to the performance of a model simultaneously. Figure 7 shows the LEAplots of features from a same correlated group, where they exhibit similar error patterns, demonstrating that these features are providing the same error information.

APPENDIX C END-TO-END MITIGATION EFFECTIVENESS

We implement the LEAF detection, explanation, and mitigation pipeline and evaluate model performance before and after LEAF mitigation. We study how LEAF's mitigation schemes improve model performance over the duration of the constant eNodeB dataset. As the vast majority of errors occur in the tail end of the distribution (see Figure 5a), we focus our study on the 95^{th} percentile of errors. Further, we analyze the average error over time to validate the impact that LEAF has on the model performance.

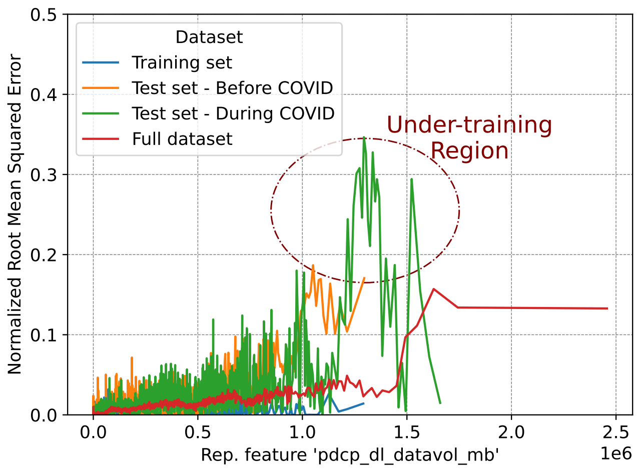
Table 7 and Figure 8, respectively, show the 95^{th} percentile of the normalized error over the dataset and the daily NRMSE performance for the CatBoost model when applied to 6 KPIs (defined in Table 2). We compare results obtained applying LEAF versus the static model (no retraining). For each target KPI, we train a static model with the best configuration obtained in Section 3, *i.e.*, 14 days of data before July 1st, 2018. We apply LEAF on this baseline. Here, we show only results

1

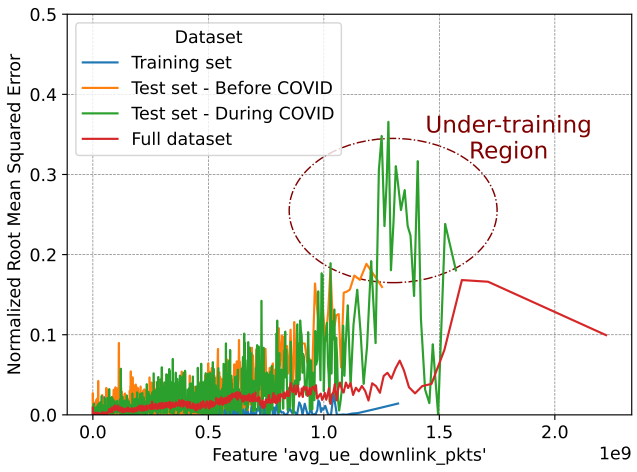
for CatBoost as the overall behavior is consistent across different models. We report detailed performance results across models and KPIs in the next subsection.

We observe in Table 7 that errors in the tail are largely mitigated using LEAF on DVol, PU, DTP, and REst. For DVol, we observe that the 95^{th} normalized error of CatBoost is effectively reduced from 0.29 for the static model to 0.19 after LEAF is applied. PU is also significantly mitigated by LEAF, reducing the 95^{th} percentile of errors from 0.86 to 0.27. However, CDR and GDR prove more difficult to mitigate possibly because that these two KPIs are highly dispersed (the coefficients of variance for them are 1.35 and 2.12 respectively, 2x to 4x higher compared to DVol, DTP, and REst). The errors obtained are only slight improvement over the baseline.

In Figure 8 we study the NRMSE performance over time. We observe that LEAF's mitigation consistently improves model performance compared to the static model. The NRMSE obtained applying LEAF for DVol forecasting (Figure 8a) never surpasses 0.125 (with the exception of spikes due to data errors) and goes as low as 0.02. This ensures a bounded error distribution and decreases the average error by 32.67%. We also observe that mitigation promptly occurs once drift is detected. In Figure 8a and Figure 8d, this is particularly evident for sudden changes in the NRMSE distribution (*e.g.*, around January and April 2020 because of COVID-19 lockdowns) and gradual changes (*e.g.*, from the second half of 2021 to the beginning of 2022). Similarly, for PU forecast we observe in Figure 8b that large unexpected errors are promptly mitigated by LEAF. High errors exceeding an NRMSE of 1 caused by missing data are observed during only a week (around June 2019), in contrast to the static model that continues to exhibit high errors for more than half a year. Finally, While LEAF still slightly improves model performance for highly dispersed KPIs like CDR (Figure 8e) and GDR (Figure 8f), they prove more difficult to mitigate. For example, due to the bursty nature of GDR, LEAF exhibits two bursty error spikes in Figure 8f, around December 2020 and August 2021, while all other KPIs do not show this pattern.



(a) *Feature group 1, the most representative feature.*

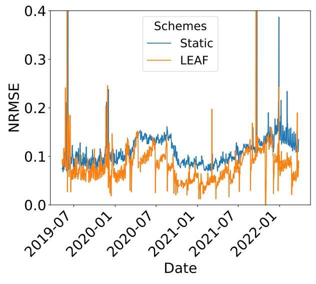


(b) *Feature group 1, the 2nd most representative feature.*

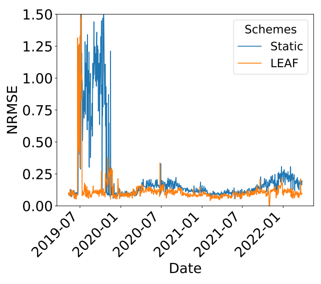
Figure 7: *The LEAplots (1,000 bins) that decomposes CatBoost NRMSE time-series in Figure 2a. The distribution of estimated local error is shown along with the values of features from the the group:* `pdcp_dl_datavol_mb` *(downlink volume in Mbps, most representative) and* `avg_ue_downlink_pkts` *(average number of downlink packets per UE, 2nd most representative).*

KPI	**95th Error**	
	Static	**LEAF**
DVol	0.29	0.19
PU	0.86	0.27
DTP	0.17	0.13
REst	0.33	0.18
CDR	0.24	0.23
GDR	0.27	0.27

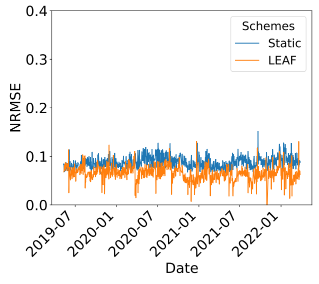
Table 7: *95th Normalized Error for CatBoost model on 6 KPIs (defined in Table 2).*



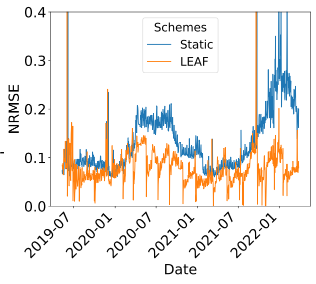
(a) *Downlink Volume*



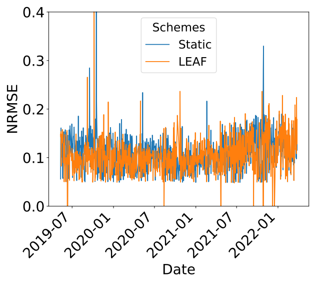
(b) *Peak active User*



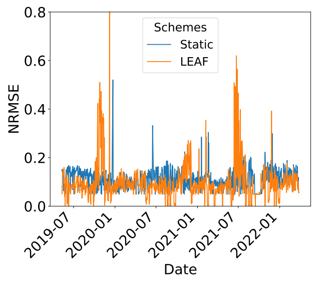
(c) *Downlink Throughput*



(d) *RRC Establishmentatt*



(e) *Call Drop Rate*



(f) *Gap Duration Ratio*

Figure 8: *NRMSE time-series before (static) and after (LEAF) mitigation, using LEAF.*