# Mitigating the Latency-Accuracy Trade-off in Mobile Data Analytics Systems

Anand Padmanabha Iyer
University of California, Berkeley
api@cs.berkeley.edu

Li Erran Li
Fudan University & Pony.ai Research Institute
erranlli@gmail.com

Mosharaf Chowdhury
University of Michigan
mosharaf@umich.edu

Ion Stoica
University of California, Berkeley
istoica@cs.berkeley.edu

## ABSTRACT

An increasing amount of mobile analytics is performed on data that is procured in a real-time fashion to make real-time decisions. Such tasks include simple reporting on streams to sophisticated model building. However, the practicality of these analyses are impeded in several domains because they are faced with a fundamental trade-off between data collection latency and analysis accuracy.

In this paper, we first study this trade-off in the context of a specific domain, Cellular Radio Access Networks (RAN). We find that the trade-off can be resolved using two broad, general techniques: intelligent data grouping and task formulations that leverage domain characteristics. Based on this, we present CellScope, a system that applies a domain specific formulation and application of Multi-task Learning (MTL) to RAN performance analysis. It uses three techniques: feature engineering to transform raw data into effective features, a PCA inspired similarity metric to group data from geographically nearby base stations sharing performance commonalities, and a hybrid online-offline model for efficient model updates. Our evaluation shows that CellScope's accuracy improvements over direct application of ML range from 2.5× to 4.4× while reducing the model update overhead by up to 4.8×. We have also used CellScope to analyze an LTE network of over 2 million subscribers, where it reduced troubleshooting efforts by several magnitudes.

We then apply the underlying techniques in CellScope to another domain specific problem, mobile phone energy bug diagnosis, and show that the techniques are general.

## CCS CONCEPTS

• **Information systems** → **Data analytics**; • **Networks** → *Mobile networks*;

## KEYWORDS

mobile systems, data analytics, cellular networks

**ACM Reference Format:**
Anand Padmanabha Iyer, Li Erran Li, Mosharaf Chowdhury, and Ion Stoica. 2018. Mitigating the Latency-Accuracy Trade-off in Mobile Data Analytics Systems. In *MobiCom '18: 24th Annual Int'l Conf. on Mobile Computing and Networking, Oct. 29–Nov. 2, 2018, New Delhi, India.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3241539.3241581

## 1 INTRODUCTION

Mobile data science and analytics has gained popularity in the recent past, with applications in diverse domains such as cellular networks [25], Internet-of-Things and machine-to-machine communication. Increasingly, the trend in such analyses has moved towards tasks that operate on data that is procured in a real-time fashion to produce low-latency decisions. Unlike traditional tasks such as aggregates or datacubes, these real-time analytic tasks often involve model building and refinement for the purpose of manual or automatic decision making. However, such analyses are faced with a fundamental trade-off between *having not enough data to build accurate-enough models in short timespans* and *waiting to collect enough data that entails stale results* in several domains. In this paper, we seek to answer the question of whether it is possible to mitigate this trade-off. Towards this goal, we take the first step and investigate this trade-off in detail, expose the effects of it, and build techniques to mitigate it in one such domain-specific problem: *performance diagnostics in cellular Radio Access Networks (RAN)s.*


*MobiCom'18, October 29–November 2, 2018, New Delhi, India*

ACM ISBN 978-1-4503-5903-0/18/10…$15.00
https://doi.org/10.1145/3241539.3241581

While RAN technologies have seen tremendous improvements over the past decade [41, 42, 46], performance problems are still prevalent [44]. Factors impacting RAN performance include user mobility, skewed traffic pattern, interference, lack of coverage, unoptimized configuration parameters, inefficient algorithms, equipment failures, software bugs and protocol errors [43]. Though some of these factors are present in traditional networks and troubleshooting these networks has received considerable attention in the literature [2, 7, 12, 53, 59], RAN performance diagnosis brings out a unique challenge: the performance of multiple base stations exhibit complex temporal and spatial interdependencies due to the shared radio access media and user mobility.

Existing systems [4, 16] for detecting performance problems rely on monitoring aggregate metrics, such as connection drop rate and throughput per cell, over minutes-long time windows. Degradation of these metrics trigger mostly manual—hence, time-consuming and error-prone—root cause analysis. Furthermore, due to their dependence on aggregate information, these tools either overlook many performance problems such as temporal spikes leading to cascading failures or are unable to isolate root causes. The challenges associated with leveraging just aggregate metrics has led operators to collect detailed traces from their network [15] to aid domain experts in diagnosing problems.

However, the sheer volume of the data and its high dimensionality make the troubleshooting using human experts and traditional rule-based systems very hard, if not infeasible [29]. Machine learning (ML) is one natural alternative to these approaches that has been used recently to troubleshoot other complex systems with considerable success. However, simply applying ML to RAN diagnosis is not enough. The desire to troubleshoot RANs as fast as possible exposes the inherent tradeoff between *latency* and *accuracy* that is shared by many ML algorithms.

To illustrate this tradeoff, consider the natural solution of building a model on a per-base station basis. On one hand, if we want to troubleshoot quickly, the amount of data collected for a given base station may not be enough to learn an accurate model. On the other hand, if we wait long enough to learn a more accurate model, this will come at the cost of delaying troubleshooting and the learned model may not be valid any longer. Another alternative would be to learn one model over the entire data set. Unfortunately, since base stations can have very different characteristics using a single model for all of them can also result in low accuracy (§2).

We present CellScope, a system that enables fast and accurate RAN performance diagnosis by resolving the *latency* and *accuracy* trade-off using two broad techniques: intelligent data grouping and task formulations that leverage domain characteristics. More specifically, CellScope applies Multi-task Learning (MTL) [11, 50], a state-of-the-art machine learning approach, to RAN troubleshooting. In a nutshell, MTL learns multiple related models in parallel by leveraging the commonality between those models. To enable the application of MTL, CellScope uses two techniques. First, it uses *feature engineering* to identify the relevant features to use for learning. Second, it uses a PCA based similarity metric to group base stations that share common features, such as interference and load. This is necessary since MTL assumes that the models have some commonality which is not necessarily the case in our setting, e.g., different base stations might exhibit different features. Note that while PCA has been traditionally used to find network anomalies, CellScope uses it for finding the common features instead.

To this end, CellScope uses MTL to create a hybrid model: an offline base model that captures common features, and an online per-base station model that captures the individual features of the base stations. This hybrid approach allows us to incrementally update the online model based on the base model. The resulting models are both accurate and fast to update. Finally, in this approach, finding anomalies is equivalent to detecting concept drift [19]. To demonstrate the effectiveness of our proposal, we have built CellScope on Spark [31, 47, 56]. Our evaluation shows that CellScope is able to achieve accuracy improvements up to 4.4× without incurring the latency overhead associated with normal approaches (§6). We have also used CellScope to analyze a live LTE network consisting of over 2 million subscribers, where we show that it could save the operator several orders of magnitude savings in troubleshooting efforts (§7).

We then investigate if the techniques we present in this paper can be general. To do so, we take a new domain-specific problem, *energy bug diagnosis in mobile phones*, and illustrate that the trade-off exists in this domain too. Using a dataset from 800,000+ users, we how the proposed techniques in CellScope can easily be adapted and demonstrate their effectiveness in mitigating the trade-off (§8).

## 2 BACKGROUND AND MOTIVATION

We begin with a brief primer on LTE networks and the current state of RAN troubleshooting. Then, we illustrate the difficulties in applying ML for RAN performance diagnosis.

### 2.1 LTE Network Primer

LTE networks provide User Equipments (UEs) such as smartphones with Internet connectivity. When a UE has data to send to or receive from the Internet, it sets up a communication channel between itself and the Packet Data Network Gateway (P-GW). This involves message exchanges between the UE and the Mobility Management Entity (MME). In coordination with the base station (eNodeB), the Serving Gateway

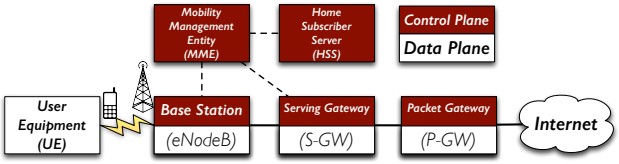



**Figure 1: LTE network architecture**

(S-GW), and P-GW, data plane (GTP) tunnels are established between the base station and the S-GW, and between the S-GW and the P-GW. Together with the connection between the UE and the base station, the network establishes a communication channel called EPS bearer (short for bearer). The LTE network architecture is shown in fig. 1.

For network access and service, LTE network entities exchange control plane messages. A specific sequence of such control plane message exchange is called a *network procedure*. For example, when a UE powers up, it initiates an *attach procedure* with the MME which consists of establishing a radio connection, authentication and resource allocation. Each network procedure involves the exchange of several messages between two or more entities. Their specifications are defined by 3GPP Technical Specification Groups (TSG) [48].

Network performance degrades and end-user experience is affected when procedure failures happen. The complex nature of these procedures (due to the multiple underlying message and entity interactions) make diagnosing problems challenging. Thus, to aid RAN troubleshooting, operators collect extensive measurements from their network. These measurements typically consist of per-procedure information (e.g., attach). To analyze a procedure failure, it is often useful to look at the associated variables. For instance, a failed attachment procedure may be diagnosed if the underlying signal strength information was captured. Hence, relevant metadata is also captured with procedure information. Since there are hundreds of procedures in the network and each procedure can have many possible metadata fields, the collected data contains several hundreds of fields.

## 2.2 RAN Troubleshooting Today

Current RAN network monitoring depends on cell-level aggregate Key Performance Indicators (KPI). Existing practice is to use performance counters to derive these KPIs. The derived KPIs are then monitored by domain experts, aggregated over certain pre-defined time window. Based on domain knowledge and operational experience, these KPIs are used to determine if service level agreements (SLA) are met. For instance, an operator may have designed the network to have no more than 0.5% call drops in a 10 minute window. When a KPI that is being monitored crosses the threshold, an alarm is raised and a ticket created. This ticket is then handled by experts who investigate the cause of the problem, often manually. Several commercial solutions exists [3–5, 16] that aid in this troubleshooting procedure by enabling efficient slicing and dicing on data. However, we have learned from domain experts that often it is desirable to apply different models or algorithms on the data for detailed diagnosis. Thus, many of the RAN trouble tickets end up with experts who work directly on the raw measurement data.

## 2.3 Machine Learning for RAN Diagnostics

The large volume of data collected in the RAN makes it an ideal candidate for the application of machine learning. We now discuss the difficulties in using ML for the purpose of RAN performance diagnostics.

*2.3.1* ***Data.*** We obtained measurement data from the live network of a top tier operator in the United States. The data consists of four types of records:

*Bearer Records:* These log bearer level information. In our logs, such information includes extensive information, such as frame loss rate, physical radio resources allocated, radio channel quality, physical layer modulation and coding rate, bearer start and end time, bearer setup delay, failure reason code (if any), associated base station, MME, S-GW and P-GW.

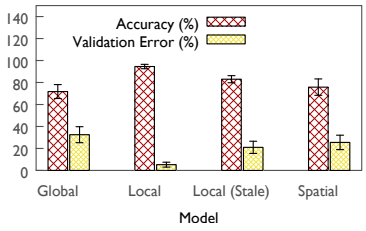
*Signaling Records:* These are logs of network procedures, such as paging, attach/detach, and handoff information. Every procedure in the network creates a new record along with metadata information such as the time of the event.

*TCP Flow Records:* These logs are from strategically placed probes in the network, and consists of TCP flow level information. They are associated with the bearer records to get more insights on application level information.

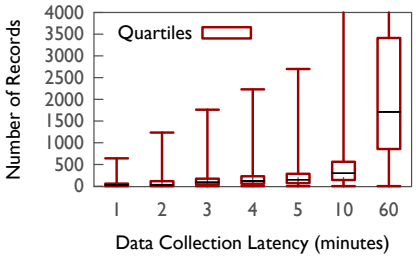
*Network Element Records:* These are aggregate information at network elements such as eNodeB or MME. Some fields in this record include total failures and downlink/uplink frames.

Collectively, the dataset contains over four hundred fields which could potentially be leveraged as individual features by a machine learning algorithm.

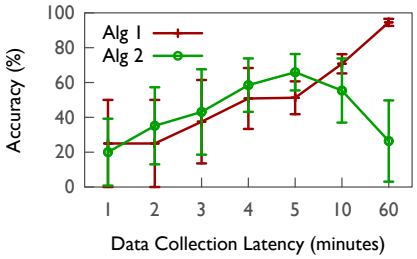
*2.3.2* ***Ineffectiveness of Global Model.*** A common approach in applying ML on a dataset is to consider the dataset as a single entity and build one model over the entire data. However, base stations in a cellular network exhibit different characteristics. This renders the use of a global model ineffective. To illustrate this problem, we conducted an experiment where the goal is to build a model for call drops in the network (similar to [26]) using information in our traces. Specifically, we build a decision tree model using an hour worth of data to ensure sufficient data for the algorithm to produce statistically significant results. Figure 2a shows the result of this experiment, where we see that the global model




**(a) A single global model results in poor accuracy. Local model is the best, but requires enough data to build and needs frequent updates, without which the staleness affects the performance. Combining geographically nearby base stations do not result in better accuracy.**




**(b) Distribution of data collected by base stations under various data collection latencies. It takes about an hour for a majority of the base stations to collect data to build statistically significant models.**




**(c) For building local models, it may not be possible to simply wait for enough data. A random forest model (Alg 1) gains from more data, but a lasso regression model (Alg2) degrades with more data due to temporal effects.**

**Figure 2: Simply applying ML for RAN performance diagnosis results in a fundamental trade-off between latency and accuracy.**

achieves poor accuracy and high variance. This underlines the heterogeneity in the characteristics of base stations and hence the ineffectiveness of global models.

*2.3.3* ***Latency/Accuracy Issues with Local Models.*** The alternative to a single global model is to build a model for every base station. We evaluate this approach by repeating the last experiment, but this time segregating the data for every base station and building an independent model for each. The results of this experiment is shown in fig. 2a, which indicates that local models are far superior, with up to 20% more accuracy while showing much lower variance.

It is natural to think of a per base station model as the final solution to this problem. However, this approach has issues too. Due to the difference in characteristics of the base stations, the amount of data they collect in a given time interval varies vastly. Thus, in small intervals, they may not generate enough data to produce statistically significant results. Figure 2b shows the distribution of the amount of data generated by these base stations under different data collection latencies. It shows that at small intervals (e.g., under 10 minutes), most base stations do not generate enough data, and that it takes about an hour for all quartiles of base stations to log reasonable number of records.

To illustrate the effect of this discrepancy, we conduct another experiment. We use two machine learning algorithms—a random forest model to predict connection drops (Alg 1), and a lasso regression model using stochastic gradient descent to predict the throughput (Alg 2)—at various data collection latencies. These two algorithms represent some of the commonly used models from the broad categories of classification and regression. The result of this experiment is shown in fig. 2c. The behavior of Alg 1 is obvious; as it gets more data its accuracy improves due to the slow varying nature of the underlying causes of failures. After an hour latency[1], it is able to reach a respectable accuracy. However, the second algorithm's accuracy initially seems to improve with more data, but falls quickly. This is counterintuitive in normal settings, but the explanation lies in the spatio-temporal characteristics of cellular networks. Many of the performance metrics exhibit high temporal variability, and thus need to be analyzed in smaller intervals. Thus, in models like that in Alg 2, it is not enough to just "wait" for enough data to be collected, and hence local modeling is ineffective.

*2.3.4* ***Need for Model Updates.*** An obvious, but flawed conclusion from our previous experiment is that models similar to that built by Alg 1 would work after the data collection latency (of an hour) has been incurred once. Put differently, can we just use historic data? In any application of ML, models need to be updated to retain their performance. This is true in cellular networks too, where temporal variations affect the performance of the model. To depict this, we repeated the experiment where we built per base station decision tree model for call drops. However, instead of training and testing on parts of the same dataset, we train on an hours worth of data, and test it on the next hour. Figure 2a shows that the accuracy drops by 12% with a stale model (because the model built using historic data is no longer valid). Thus, it is important to keep the model fresh by incorporating incoming data and removing old data. Such sliding updates to ML models in a general setting is difficult due to the overheads in retraining them from scratch. To add to this, cellular networks consist of several thousands of base stations. Thus, a per base station approach requires creating and updating a huge amount of models (e.g., our network consisted of over 13000 base stations). This makes scaling hard.

*2.3.5* ***Why not Spatial/Temporal Partitioning?*** Our experiments point towards the need for obtaining enough

[1]Such high latencies may not be acceptable in many scenarios.

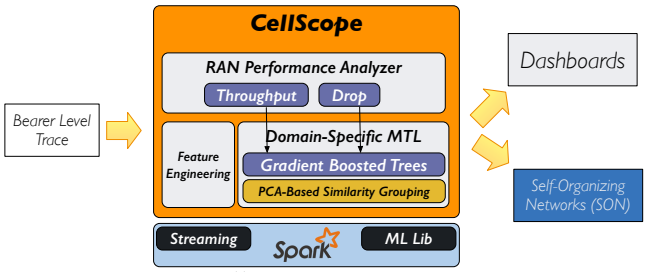



**Figure 3: CellScope System Architecture.**

data for ML algorithms to produce statistically significant results with low latency. The obvious solution to combating this trade-off is to *intelligently* combine data from multiple base stations. It is intuitive to think of this as a spatial partitioning problem, since base stations in the real world are geographically separated. Thus, it is reasonable to assume that a spatial partitioner which combines data from base stations within a geographical region must be able to give good results. Unfortunately, this isn't the case which we motivate using a simple example. Consider two base stations, one situated at the center of times square in New York and the other a mile away at a residential area. Using a spatial partitioning scheme that divides the space into equal sized planes would likely result in combining data from these base stations. However, this is not desirable because of the difference in characteristics of these base stations[2]. We illustrate this using the drop modeling experiment as before. Figure 2a shows the performance where we combine data from nearby base stations using a simple grid partitioner, and then build a model in each of the partitions. The result shows that this technique is only slightly better compared to a single global model. We evaluate other spatial partitioning schemes in §6.

# 3 CELLSCOPE OVERVIEW

We now present our solution, CellScope, which mitigates the latency-accuracy trade-off using a domain-specific formulation and application of Multi-Task Learning (MTL).

## 3.1 Problem Statement

CellScope's ultimate goal is to enable *fast and accurate RAN performance diagnosis by resolving the trade-off between data collection latency and the achieved accuracy*. The key difficulty arises from the fundamental trade-off between *having not enough data to build accurate-enough models in short timespans* and *waiting to collect enough data that entails stale results*. Additionally, CellScope must also support efficient modifications to the learned models to account for the temporal nature of our setting to avoid data and model staleness.

[2]In our measurements, a base station in a highly popular spot serves more than 300 UEs and carries multiple times uplink / downlink traffic compared to another base station situated just a mile from it that serves only 50 UEs.

## 3.2 Architectural Overview

Figure 3 shows the high-level architecture of CellScope, which consists of the following key components:

**Input data:** CellScope uses measurement traces that are readily available in modern cellular networks (§2.1). Base stations collect traces independently and send them to the associated MME, which merges records if required and uploads them to a data center.[3]

**Feature engineering:** Next, CellScope uses domain knowledge to transform the raw data and construct a set of features amenable to learning (e.g., computing interference ratios)(§4.1). We also leverage protocol details and algorithms (e.g., link adaptation in the physical layer).

**Domain-specific MTL:** CellScope uses a domain specific formulation and application of MTL that allows it to perform accurate diagnosis while updating models efficiently (§4.2).

**Data partitioner:** To enable correct application of MTL, CellScope implements a partitioner based on a similarity score derived from Principal Component Analysis (PCA) and geographical distance (§4.3). The partitioner segregates data to be analyzed into independent sets and produces a smaller co-located set relevant to the underlying analysis. This also minimizes the need to shuffle data during training.

**RAN performance analyzer:** This component binds everything together to build diagnosis modules. It leverages the MTL component and uses appropriate techniques to build call drop and throughput models. We discuss our experience of applying these techniques to a live LTE network in §7. This component can be easily replaced to extend CellScope to a new domain, as we show in §8.

**Output:** Finally, CellScope can output analytics results to external modules such as RAN performance dashboards. It can also provide inputs to Self-Organizing Networks (SON).

# 4 MITIGATING LATENCY ACCURACY TRADE-OFF

In this section, we present how CellScope mitigates the trade-off between latency and accuracy. We first discuss a high-level overview of RAN specific feature engineering that prepares the data for learning (§ 4.1). Next, we describe CellScope's MTL formulation (§ 4.2), and how it lets us build fast, accurate and incremental models. Then, we explain how CellScope achieves grouping that captures commonalities among base stations using a novel PCA based partitioner that makes application of MTL possible (§ 4.3).

## 4.1 Feature Engineering

Feature engineering, the process of transforming the raw input data to a set of features that can be effectively utilized

[3]The transfer of traces to a data center is not fundamental. Extending CellScope to do geo-distributed learning in a future work.

by machine learning algorithms, is a fundamental part of ML applications [58]. Generally carried out by domain experts, it is often the first step in ML.

In CellScope, the network measurement data contains several hundreds of fields (§2). These fields range from simple bearer identification information to fields associated with LTE network procedures. Unfortunately, many of these fields are not suitable for model building as it is. Additionally, several fields are collected in a format that utilizes a compact representation. Finally, these records are not self-contained, and multiple records may need to be "joined" to create a feature for a certain procedure. We use simple feature engineering to obtain fields that can be used in ML algorithms. As an example, for modeling connection drop rates, we use block error rate (BLER) as a feature. However, the records do not directly provide this value, thus it is computed using the block transfer information. Similarly, for throughput modeling, the downlink and uplink throughput values are computed using the amount of physical resource blocks allocated and the transfer time. While we depend on manual feature engineering in this work (automating this is part of our future work), not all fields need to be feature engineered. Further, we found that the engineered fields can be used across several ML algorithms.

## 4.2 Multi-Task Learning

The latency-accuracy trade-off makes it hard to achieve both low latency and high accuracy in ML tasks (§2). The ideal-case scenario in CellScope is if infinite amount of data is available per base station with zero latency. In this scenario, we would have a learning task for each base station that produce a model as an output with the best achievable accuracy. In reality, our setting has several tasks, each with its own data. However, each task does not have enough data to produce models with acceptable accuracy in a given latency budget. This makes our setting an ideal candidate for multi-task learning (MTL), a research area in machine learning that has been successful in many ML applications. The key idea behind MTL is to *learn from other tasks by weakly coupling their parameters so that the statistical efficiency of many tasks can be boosted* [10, 11, 17, 50]. Specifically, if we are interested in building a model of the form

$$h(x) = m(f_1(x), f_2(x), ..., f_k(x)) \quad (1)$$

where $m$ is a model (e.g., to predict connection drop) composed of feature functions $f_1$ through $f_k$, then the traditional MTL formulation, given dataset $\mathcal{D} = \{(x_i, y_i, bs_i) : i = 1, ..., n\}$, where $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$ and $bs_i$ denotes the $i^{th}$ base station, is to learn

$$h(x) = m_{bs}(f_1(x), f_2(x), ..., f_k(x)) \quad (2)$$

where $m_{bs}$ is a per base station model.

In this MTL formulation, the core assumption is a shared structure or dependency across each of the learning problems. Unfortunately, in our setting, the base stations do not share a structure at a global level (§2). Due to their geographic separation and the complexities of wireless signal propagation, the base stations share a spatio-temporal structure instead. Thus, we proposes a new domain-specific MTL formulation.

*4.2.1* ***CellScope's MTL Formulation.*** In order to address the difficulty in applying MTL due to the violation of task dependency assumption in RANs, we can leverage domain-specific characteristics. Although independent learning tasks (learning per base station) are not correlated with each other, they exhibit specific non-random structure. For example, the performance characteristics of base stations nearby are influenced by similar underlying features. Thus, we propose exploiting this knowledge to segregate learning tasks into groups of dependent tasks on which MTL can be applied. MTL in the face of dependency violation has been studied in the machine learning literature in the recent past [20, 30]. However, they assume that each group has its own set of features. This is not entirely true in our setting, where multiple groups may share most or all features but still need to be treated as separate groups. Furthermore, some of the techniques used for automatic grouping without a priori knowledge are computationally intensive.

Assuming we can club learning tasks into groups, we can rewrite the MTL equation in eq. (2) as:

$$h(x) = m_{g(bs)}(f_1(x), f_2(x), ..., f_k(x)) \quad (3)$$

where $m_{g(bs)}$ is the per-base station model in group $g$. We describe a simple technique to achieve this grouping based on domain knowledge in § 4.3 and experimentally show that just grouping can achieve significant gains in §6.

In theory, the MTL formulation in eq. (3) should suffice for our purposes as it would perform much better by capturing the inter-task dependencies using grouping. However, this formulation still builds an independent model for each base station. Building and managing a large amount of models leads to significant performance overhead and would impede our goal of scalability. Scalable application of MTL in a general setting is an active area of research in machine learning [36], so we turn to problem-specific optimizations to address this challenge.

The model $m_{g(bs)}$ could be built using any class of learning functions. In this paper, we restrict ourselves to functions of the form $F(x) = w \cdot x$ where $w$ is the weight vector associated with a set of features $x$. This simple class of function gives us tremendous leverage in using standard algorithms that can easily be applied in a distributed setting, thus addressing the scalability issue. In addition to scalable model building, we must also be able to update the built models fast. However,

machine learning models are typically hard to update in real time. To address this challenge, we discuss a hybrid approach to building the models in our MTL setting next.

*4.2.2* ***Hybrid Modeling for Fast Model Updates.*** Estimation of the model in eq. (3) could be posed as an $\ell_1$ regularized loss minimization problem [51]:

$$min \sum L(h(x : f_{bs}), y) + \lambda ||R(x : f_{bs})|| \tag{4}$$

where $L(h(x : f_{bs}), y)$ is a non-negative loss function composed of parameters for a particular base station, hence capturing the error in the prediction for it in the group, and $\lambda > 0$ is a regularization parameter scaling the penalty $R(x : f_{bs})$ for the base station. However, the temporal and streaming nature of the data means that the model must be refined frequently for minimizing staleness.

Fortunately, grouping provides us an opportunity to solve this. Since the base stations are grouped into correlated task clusters, we can decompose the features used for each base station into a *shared common set* $f_c$ and a *base station specific* set $f_s$. Thus, we can modify eq. (4) as minimizing

$$\sum \left( \sum L(h(x : f_s, f_c), y) + \lambda ||R(x : f_s)|| \right) + \lambda ||R(x : f_c)|| \tag{5}$$

where the inner summation is over dataset specific to each base station. This separation gives us a powerful advantage. Since we grouped base stations, the feature set $f_s$ is minimal, and in most cases just a weight vector on the common feature set. Because the core common features do not change often, we need to update only the base station-specific parts in the model frequently, while the common set can be reused. Thus, we end up with a hybrid offline-online model. Furthermore, the choice of our learning functions lets us apply stochastic methods [45] which can be efficiently parallelized.

*4.2.3* ***Anomaly Detection Using Concept Drift.*** A common use case of learning tasks for RAN performance analysis is in detecting anomalies. For instance, an operator may be interested in learning if there is a sudden increase in call drops. At the simplest level, it is easy to answer this question by monitoring the number of call drops at each base station. However, just a yes or no answer to such questions are seldom useful. If there is a sudden increase in drops, then it is useful to understand if the issue affects a complete region and its root cause.

Our MTL approach and the ability to do fast incremental learning enables a better solution for anomaly detection and diagnosis. Concept drift is a term used to refer the phenomenon where the underlying distribution of the training data for a machine learning model changes [19]. CellScope leverages this to detect anomalies as concept drifts and proposes a simple technique for it. Since we process incoming data in mini-batches (§5), each batch can be tested quickly on the existing model for significant accuracy drops. An anomaly occurring just at a single base station would be detected by one model, while one affecting a larger area would be detected by many. Once anomaly is detected, finding cause is as easy as updating the model and comparing it with the old.

## 4.3 Data Grouping for MTL

Having discussed CellScope's MTL formulation, we now turn our focus towards how CellScope achieves efficient grouping of cellular datasets that enables accurate learning. Our data partitioning is based on Principal Component Analysis (PCA), a widely used technique in multivariate analysis [37]. PCA uses an orthogonal coordinate transformation to map a given set of points into a new coordinate space. Each of the new subspaces are commonly referred to as a principal component. Since the coordinate space is smaller than the original , PCA is used for dimensionality reduction.

In their pioneering work, Lakhina et.al. [33] showed the usefulness of PCA for network anomaly detection. They observed that it is possible to segregate normal behavior and abnormal (anomalous) behavior using PCA—the principal components explain most of the normal behavior while the anomalies are captured by the remaining subspaces. Thus, by filtering normal behavior, it is possible to find anomalies that may be otherwise undetected.

While the most common usecase for PCA has been dimensionality reduction (in machine learning domains) or anomaly detection (in networking domain), we use it in a novel way, to enable grouping of datasets for multi-task learning. Due to the lack of the ability to collect sufficient amount of data from individual base stations, detecting anomalies in them will not yield results. However, the data would still yield an explanation of normal behavior. We use this observation to partition the dataset.

*4.3.1* ***Notation.*** As bearer level traces are collected continuously, we consider a buffer of bearers as a *measurement matrix A*. Thus, $A$ consists of $m$ bearer records, each having $n$ observed parameters making it an $m \times n$ time-series matrix. It is to be noted that $n$ is in the order of a few 100 fields, while $m$ can be much higher depending on how long the buffering interval is. We enforce $n$ to be fixed in our setting—every measurement matrix must contain $n$ columns. To make this matrix amenable to PCA analysis, we adjust the columns to have zero mean. By applying PCA to any measurement matrix $A$, we can obtain a set of $k$ principal components ordered by amount of data variance they capture.

*4.3.2* ***PCA Similarity.*** It is intuitive to see that many measurement matrices may be formed based on different

criteria. Suppose we are interested in finding if two measurement matrices are *similar*. One way to achieve this is to compare the principal components of the two matrices. Krzanowski [32] describes such a *Similarity Factor* (*SF*). Consider two matrices $A$ and $B$ having the same number of columns, but not rows. The similarity factor between $A$ and $B$ is:

$$SF = trace(LM'ML') = \sum_{i=1}^{k}\sum_{j=1}^{k}\cos^2\theta_{ij}$$

where $L, M$ are the first $k$ principal components of $A$ and $B$ respectively, and $\theta_{ij}$ is the angle between the $i^{th}$ component of $A$ and the $j^{th}$ component of $B$. The similarity factor considers all combinations of $k$ components from both matrices.

*4.3.3* ***CellScope's Similarity Metric.*** Similarity in our setting bears a slightly different meaning: we do not want strict similarity between measurement matrices, but only similarity between corresponding principal components. This ensures that algorithms will still capture the underlying major influences and trends in observation sets that are not exactly similar. So we propose a simpler metric.

Consider two measurement matrices $A$ and $B$ as before, where $A$ is of size $m_A \times n$ and $B$ is of size $m_B \times n$. By applying PCA on the matrices, we can obtain $k$ principal components using a heuristic. We obtain the first $k$ components which capture 95% of the variance. From the PCA, we obtain the resulting weight vector, or *loading*, which is a $n \times k$ matrix: for each principal component in $k$, the loading describes the weight on the original $n$ features. Intuitively, this can be seen as a rough measure of the influence of each of the $n$ features on the principal components. The Euclidean distance between the corresponding loading matrices gives

$$SF_{CellScope} = \sum_{i=1}^{k} d(a_i, b_i) = \sum_{i=1}^{k}\sum_{j=1}^{n}|a_{ij} - b_{ij}|$$

where $a$ and $b$ are the column vectors representing the loadings for the corresponding principal components from $A$ and $B$. Thus, $SF_{CellScope}$ captures how closely the underlying features explain the variation in the data.

Due to the complex interactions between network components and the wireless medium, many of the performance issues in RANs are geographically tied (e.g., congestion might happen in nearby areas, and drops might be concentrated)[4]. However, $SF_{CellScope}$ doesn't capture this phenomenon because it only considers similarity in normal behavior. Consequently, it is possible for anomaly detection algorithms to miss geographically-relevant anomalies. To account for this domain-specific characteristic, we augment our similarity metric to also capture the geographical closeness by weighing the metric by geographical distance between the two measurement matrices. Our final similarity metric is[5]

$$SF_{CellScope} = w_{distance_{(A,B)}} \times \sum_{i=1}^{k}\sum_{j=1}^{n}|a_{ij} - b_{ij}|$$

*4.3.4* ***Using Similarity Metric for Partitioning.*** With similarity metric, CellScope can now partition bearer records. We first group the bearers into measurement matrices by segregating them based on the cell on which the bearer originated. The grouping is based on our observation that the cell is the lowest level at which an anomaly would manifest. We then create a graph $G(V, E)$ where the vertices are the individual cell measurement matrices. An edge is drawn between two matrices if the $SF_{CellScope}$ between them is below a threshold. To compute $SF_{CellScope}$, we simply use the geographical distance between the cells as the weight. Once the graph has been created, we run connected components on this graph to obtain the partitions. The use of connected component algorithm is not fundamental, it is also possible to use a clustering algorithm instead. For instance, a k-means clustering algorithm that could leverage $SF_{CellScope}$ to merge clusters would yield similar results.

*4.3.5* ***Managing Partitions Over Time.*** One important consideration is managing group changes over time. To detect group changes, it is necessary to establish correspondence between groups across time intervals. Once this correspondence is established, CellScope's hybrid modeling makes it easy to accommodate changes. Due to the segregation of our model into common and base station specific components, small changes to the group do not affect the common model. In these cases, we can simply bootstrap the new base station using the common model, and then start learning specific features. On the other hand, if there are significant changes to a group, then the common model may no longer be valid, which is easy to detect using concept drift. In such cases, the offline model could be rebuilt.

## 4.4 Summary

We now summarize how CellScope resolves the fundamental trade-off between latency and accuracy. To cope with the fact that individual base stations cannot produce enough data for learning in a given time budget, CellScope uses MTL. However, our datasets violate the assumption of learning task dependencies. As a solution, we proposed a novel way of using PCA to group data into sets with the same underlying performance characteristics. Directly applying MTL on these

[4]Proposals for conducting geographically weighted PCA (GW-PCA) exist [22], but they are not applicable since they assume a smooth decaying user provided bandwidth function.

[5]A similarity measure for multivariate time series is proposed in [55], but it is not applicable due to its stricter form and dependence on finding the right eigenvector matrices to extend the Frobenius norm.

groups would still be problematic in our setting due to the inefficiencies with model updates. To solve this, we proposed a new formulation for MTL which divides the model into an offline and online hybrid. On this formulation, we proposed using simple learning functions are amenable to incremental and distributed execution. Finally, CellScope uses a simple concept drift detection to find and diagnose anomalies.

# 5 IMPLEMENTATION

We have implemented CellScope on Spark [56]. We describe its API that exposes our commonality based grouping based on PCA (§ 5.1), and implementation details on the hybrid offline-online MTL models (§ 5.2).

## 5.1 Data Grouping API

CellScope's grouping API is built on Spark Streaming [57], since the data arrives continuously, and we need to operate on this data in a streaming fashion. Spark Streaming already provides support for windowing functions on streams of data, thus we extended it with the three APIs in listing 1.

```
grouped = DStream.groupBySimilarityAndWindow(
    windowDuration, slideDuration)

reduced = DStream.reduceBySimilarityAndWindow(
    func, windowDuration, slideDuration)

joined = DStream.joinBySimilarityAndWindow(
    windowDuration, slideDuration)
```

**Listing 1: Groping API**

The APIs leverage the `DStream` abstraction provided by Spark Streaming. `groupBySimilarityAndWindow` takes the buffered data from the last window duration, applies the similarity metric to produce outputs of grouped datasets every slide duration. `reduceBySimilarityAndWindow` allows an additional user defined associative reduction operation on the grouped datasets. Finally, `joinBySimilarityAndWindow` joins multiple streams using similarity.

## 5.2 Hybrid MTL Modeling

We use Spark's machine learning library, MLlib [47] for implementing our hybrid MTL model. MLlib contains implementation for many distributed learning algorithms. The MTL formulation we presented in § 4.2 allows us to utilize these existing models in our framework.

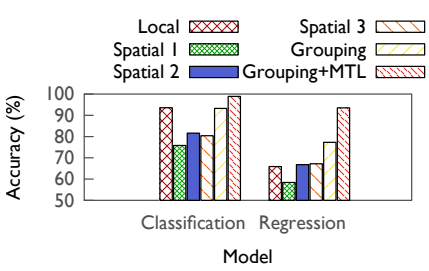
In MTL, the tasks learn from each other. These tasks in our setting consist of building a model, $m_{g(bs)}$ for each base station in every group created by the PCA based grouping. In eq. (3), we presented our MTL formulation, and described a simplified loss minimization method to estimate this model. Further, in eq. (5), we decomposed this into shared and base station specific set, so the model $m_{g(bs)}$ is of the general form $h(x : f_s, f_c), y)$. Since we restrict ourselves to learning functions of the form $w \cdot x$ for this model, our model per base station is simply a weight vector on the shared group model. This allows the usage of existing ensemble methods [14]. Ensemble methods use multiple learning algorithms to obtain better performance. In our case, we use the ensemble method to learn the shared group model. This can be done in many ways: we can directly employ existing ensemble methods, or we can leverage multiple algorithms to be components of the ensemble. However, unlike normal ensemble methods where the output is aggregated, we use the MTL approach of a task per base station to learn the per-base station model. This is equivalent to a linear model on the individual ensemble components, which gives us the weight vector.

We modified the MLLib implementation of Gradient Boosted Tree (GBT) [18]. This implementation supports both classification and regression, and internally uses stochastic methods. We implement the group's shared feature model using either the GBT's ensemble, or individual algorithms. As an example, for connection drop prediction, the shared model can be obtained using the standard ensembles such as the GBT itself, or random forests. Then, we use individual base station data to fit a linear model on the individual ensemble components. Note that it is not necessary to build the base model this way—we could also use multiple learning methods as ensemble components. In the same example, our ensemble could consist of a combination of SVM and decision trees. Similarly, for throughput prediction, the shared model is built as an ensemble of regression models—for instance, we may use one model for low throughput and another for high throughput, and each of these tasks could use a different standard learning method. In this method, we can update the base station specific weight vector in real time as data is streamed in, as we simply need to update the linear model. Further, the group specific model can be periodically retrained. One way to do so is to simply add more models to the ensemble when new data comes in. Our implementation allows weighing the outcome to give more weights to the latest models.

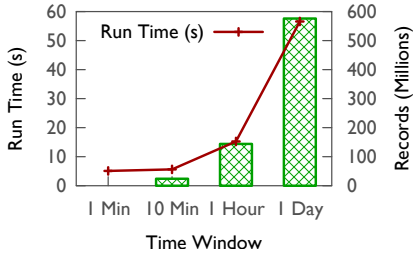
# 6 EVALUATION

We have evaluated CellScope through a series of experiments on real-world cellular traces from a live LTE network in a large geographical area. Our results show that:

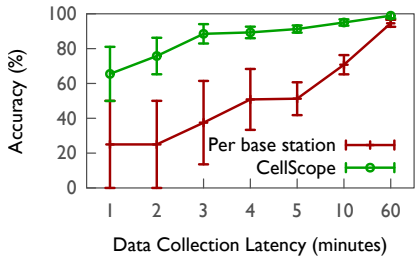
- CellScope's similarity based grouping provides up to 10% improvement in accuracy on its own compared to the best space partitioning scheme.
- With MTL, CellScope's accuracy improvements range from 2.5× to 4.4× over different collection latencies.
- Our hybrid online-offline model is able to reduce model update times upto 4.8× and is able to learn changes in an online fashion with no loss in accuracy.



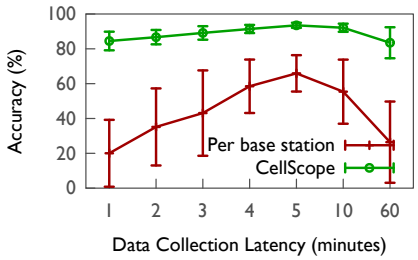

(a) Grouping by itself is able to provide significant gains. MTL provides further gains.



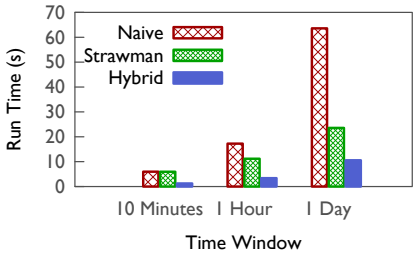

(b) Grouping is not computationally intensive, even a days worth of data (with >500M records) can be easily grouped in under a minute.



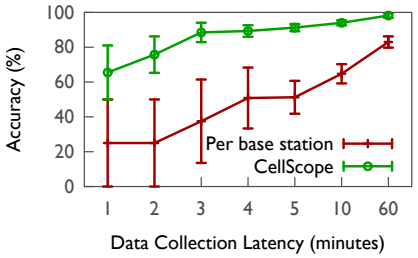

(c) CellScope achieves up to 2.5× accuracy improvements in drop rate classification.




(d) Improvements in throughput regression go up to 4.4×




(e) CellScope's hyrid model allows efficient updates, and reduces update time by up to 4.8×.




(f) Online training due to the hybrid model helps avoid the loss in accuracy due to staleness of the model.

**Figure 4: CellScope is able to achieve high accuracy while reducing the data collection latency.**

**Evaluation Setup:** We use a private cluster of 20 machines, each consisting of 4 CPUs, 32GB RAM and 200GB hard disk. **Dataset:** We collected data from a major metro-area LTE network for a time period of over 10 months. It serves over 2 million active users and carries over 6TB traffic per hour.

## 6.1 Benefits of Similarity Based Grouping

We first attempt to answer the question *"How much benefits do the similarity based grouping provide?"*. For this, we conducted two experiments, each with a different learning algorithm. The first experiment, detection of call drops, uses a classification algorithm while the second, throughput prediction, uses a regression algorithm. We chose these to evaluate the benefits in two different classes of algorithms. In both these cases, we pick the data collection latency where the per base station model gives the best accuracy, which was 1 hour for classification and 5 minutes for regression. In order to compare the benefits of our grouping scheme alone, we build a single model per group instead of applying MTL. We compare the accuracy obtained with three different space partitioning schemes. The first scheme (Spatial 1) just partitions space into grids of equal size. The second (Spatial 2) uses a sophisticated space-filling curve based approach [25] that could create dynamically sized partitions. Finally, the third (Spatial 3) creates partitions using base stations that are under the same region. Figure 4a shows the results.

CellScope's similarity grouping performs as good as the per base station model which gives the highest accuracy. It is interesting to note the performance of spatial partitioning schemes which ranges from 75% to 80%. None of the spatial schemes come close to the similarity grouping results. This is because the drops are few, and concentrated. Spatial schemes club base stations not based on underlying drop characteristics, but only based on spatial proximity. This causes the algorithms to underfit or overfit. Since our similarity based partitioner groups base stations using the drop characteristics, it is able to do as much as 17% better.

The benefits are even higher in the regression case. Here, the per base station model is unable to get enough data to build an accurate model and hence is only able to achieve around 66% accuracy. Spatial schemes are able to do slightly better than that. Our similarity based grouping emerges as a clear winner in this case with 77.3% accuracy. This result depicts the highly variable performance characteristics of the base stations, and the need to capture them for accuracy. These benefits do not come at the cost of computational overhead due to grouping. Figure 4b shows the overhead of similarity based grouping on various dataset sizes.

## 6.2 Benefits of MTL

Next, we characterize the benefits of CellScope's use of MTL. We repeated the experiment before, and apply MTL to the

grouped data to see if the accuracy improves compared to the earlier approach of a single model per group. The results are presented in figure 4a. The ability of MTL to learn and improve models from other similar base stations' data results in an increase in the accuracy. Over the benefits of grouping, we see an improvement of 6% in the connection drop diagnosis experiment, and 16.2% in the case of throughput prediction experiment. The higher benefits in the latter comes from CellScope's ability to capture individual characteristics of the base station. This ability is not so crucial in the former because of the limited variation in individual characteristics.

### 6.3 Combined Benefits

Here, we are interested in evaluating how CellScope handles the latency accuracy trade-off. We do the same classification and regression experiments, but on different data collection latencies. We show the results from the classification and regression experiment in fig. 4c and fig. 4d, which compares CellScope's accuracy against a per base station model's.

When the opportunity to collect data at individual base stations is limited, CellScope is able to leverage our MTL formulation to combine data from multiple base stations, and build customized models to improve the accuracy. The benefits of CellScope ranges up to 2.5× in the classification experiment, to 4.4× in the regression experiment. Lower latencies are problematic in the classification experiment due to the extremely low probability of drops, while higher latencies are a problem in the regression experiment due to the temporal changes in performance.

### 6.4 Hybrid model benefits

Finally, we are interested in learning how much overhead it reduces during model updates, and if it do online learning.

To answer the first question, we conducted the following experiment: we considered three different data collection latencies: 10 minute, 1 hour and 1 day. We then learn a decision tree model on this data in a tumbling window fashion. So for the 10 minute latency, we collect data for 10 minutes, then build a model, wait another 10 minutes to refine the model and so on. We compare our hybrid model strategy to two different strategies: a naive approach which rebuilds the model from scratch every time, and a better, strawman approach which reuses the last model, and makes changes to it. Both builds a single model while CellScope uses our hybrid MTL model and only updates the online part of the model. The results of this experiment is shown in figure 4e.

The naive approach incurs the highest overhead, which is obvious due to the need to rebuild the entire model from scratch. The overhead increases with the increase in input data. The strawman approach, on the other hand, is able to avoid this heavy overhead. However, it still incurs overheads with larger input because of its use of a single model which requires changes to many parts of the tree. CellScope incurs the least overhead, due to its use of multiple models. When data accumulates, it only needs to update a part of an existing tree, or build a new tree. This strategy results in a reduction of up to 2.2× to 4.8× in model building time for CellScope.

To wrap up, we evaluated the performance of the hybrid strategy on different data collection intervals. Here we are interested in seeing if the hybrid model is able to adapt to data changes and provide reasonable accuracies. We use the connection drop experiment again, but do it in a different way. At different collection latencies, we build the model at the beginning of the collection and use the model for the next interval. Hence, for the 1 minute latency, we build a model using the first minute data, and use the model for the second minute (until the whole second minute has arrived). The results are shown in figure 4f. We see here that the per base station model suffers an accuracy loss at higher latencies due to staleness, while CellScope incurs almost zero loss in accuracy. This is because it doesn't wait until the end of the interval, and is able to incorporate data in real time.

## 7 REAL WORLD RAN ANALYSIS

We now turn to the question of how could operators benefit from a system such as CellScope? We try to answer this question in two ways: first, we try to evaluate what are the benefits of automatic root-causing and how much effort is reduced for the operator because of this feature. Second, we evaluate CellScope's ability to analyze in the wild.

### 7.1 Time Savings to the Operator

Operators spend several billions of dollars in diagnosing network problems. Often, finding the cause of a network problem takes hours, or even days of effort. To evaluate how CellScope could cut down this effort, we collected network trouble tickets from the operator. The operator logs tickets at different levels, so we look at trouble tickets that were investigated by domain experts using state-of-the-art tools such as datacubes. For each ticket where the operator has network data available, we used CellScope to diagnose the problem. This way, we can evaluate the potential time savings CellScope provides. We discuss four real trouble tickets, the time taken by CellScope is depicted in table 1.

*7.1.1 Throughput Degradation After Upgrade.* This ticket reported that a number of users experienced degraded network throughput after a network upgrade. In many cases, throughput decrease of up to 30% was observed. Since not all of the users saw this problem, the operator had to conduct field trials to find the root cause of the problem. We used CellScope to model the throughput **before** and **after** the upgrade. Comparing the models, we noticed that a cluster of

| *Ticket* | *Resolution Time* | *CellScope* |
|---|---|---|
| § 7.1.1 | 3 days | 10 minutes |
| § 7.1.2 | 1 day | 2 minutes |
| § 7.1.3 | 7 days | 15 minutes |
| § 7.1.4 | 1 hour | 1 minute |

**Table 1: CellScope is able to reduce operator effort by several orders of magnitude. Resolution time includes field trials & expert analysis using datacubes / state-of-the-art tools [3].**

base stations had one feature influencing the model heavily. This matched the operator's ticket resolution—the field trials in the ticket indicated that the problem was cluster-wise and that it was because the feature CellScope was erroneously turned on after the upgrade. The base stations CellScope identified matched those reported in the resolution. In this case, the ticket was resolved in three days including the field trials, while our modeling on CellScope took less than 10 minutes. Note that manually applying learning techniques would not have found the problem without grouping.

*7.1.2 Specific Patterns of Call Drops .* Here, the operator reported consistent call drops (specifically, VoLTE call drops) in certain areas of the network. Manually analyzing this would have required a domain expert to slice and dice several TB of data to find a pattern and then dig deep into the pattern. To reduce this effort, the operator conducted field trials in parts affected to obtain test data that is manageable for the expert, who was able to identify the problem: a missing neighbor configuration in a group of base stations.

We used CellScope to model the call drop in an expanded portion of the network. After the grouping process, one particular group's model indicated that drops happened when a handoff procedure was triggered and the procedure failed due to a specific error code at the base station, `missing-neighbor`. Here, the field trial, and domain expert's analysis was completed in one business day, while CellScope did the grouping and modeling on one day's data in 2 minutes.

*7.1.3 Periodic Throughput KPI Degradation.* The operator noticed a degradation of KPI in the network. The degradation happened in some serving cells. However, this was not consistently noticed, and occurred irregularly. To add, the problem was transient. Thus, the ticket required a week worth of effort to diagnose since field trials did not prove to be of help. We used CellScope in the following fashion: we replayed the data for days when KPI degradation was reported. We then built incremental models for drop rate and throughput. We then look at the intervals when CellScope refines the model due to accuracy loss using the *concept drift* and look at the model changes. We noticed that during some specific intervals, call drops spiked in some cells while the throughput of the entire cell dropped. The difference in the models built by CellScope indicated that device specific features influenced the drops. The reason was that a particular model and software version of a device creating a deluge of control messages that affected the entire cell when it was near capacity. The ticket closure confirmed this.

*7.1.4 Periodic Call Drops.* Here, the operator noticed periodic increase in call drops. The domain expert was able to identify the problem in an hour as PCI collision due to her vast expertise in the domain by looking through the logs from affected period. We used the same logs in CellScope, and were able to generate a call drop model that explained the drops using inter-cell interference. When expertise is not available, the ticket would have been time-consuming.

## 7.2 Analysis in the Wild: Findings

To validate our system in the real world, we used CellScope for RAN performance analysis on the live LTE network. Based on our experience with trouble tickets, we considered two metrics that are of significant importance for end-user experience: *throughput* and *connection drops*. In this section, we present some of our main findings (which were previously unknown to the operator) and the role played by CellScope.

*7.2.1 SINR Anomaly.* In a particular week, we noticed that an implementation of a learning task for connection drop predicted unusually high numbers of drops. These high drops happened at some base stations, all of which were assigned to the same group in CellScope's grouping. Upon further investigation with help from network experts, it was revealed that these base stations had been experiencing unusually higher levels of interference.

*7.2.2 Incorrect Parameters.* Similarly, we implemented a throughput prediction model. During a month long observation, we noticed that the predicted throughput for a set of base stations had fallen below its normal average after a certain date. It was found that the base stations were connected to the same MME and that a software upgrade had set some parameters affecting the throughput incorrectly. This was one of several misconfigurations we found in the network that caused performance degradation. Others included incorrect neighbor assignments and hand-off problems.

*7.2.3 Real-time Monitoring.* We simulated real-time monitoring of the network and CellScope's ability to detect performance problems. The current approach taken by the operator is to define SLAs for KPIs and then monitor them for SLA violations. However, such aggregate metrics are likely to miss many events. We used CellScope to monitor the network over a month, and verify if the events predicted by CellScope matches ground truth. Not only did CellScope detect 100% of the KPI SLA violations, it also found a few

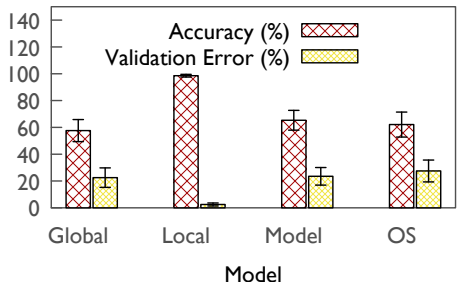



**Figure 5: Other domains suffer from latency-accuracy trade-off. Here, we see the problem in the domain of energy debugging for mobile devices. Grouping by phone model or phone operating system does not give benefits.**

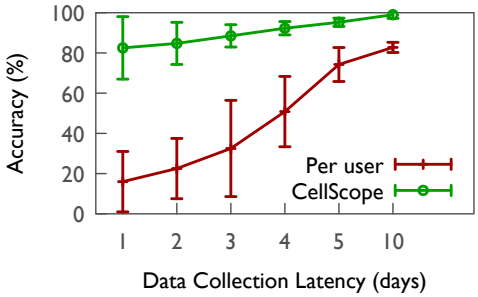



**Figure 6: CellScope's techniques can easily be extended to new domains, and can benefit them. Here, using our techniques, models built are usable immediately while without CellScope, Carat [35] takes more than a week to build a model that is usable.**

issues that were missed by the KPI based monitoring system, and later logged as trouble tickets.

*7.2.4 Measurement Error.* We also found problems in network measurements. Specifically, during initial deployment trials of CellScope, we noticed that using the feature engineered field of block error rate resulted in poor accuracy. The reason for this was an uninitialized field in the measurement record logger, which resulted in random values.

## 8 EXTENDING CELLSCOPE TO A NEW DOMAIN

To show the generality of the techniques presented in this paper, we now apply these techniques to a new domain: *energy anomaly detection in mobile phones* [35]. We obtained a dataset of measurements from approximately 800,000 users obtained using the Carat app. The goal here is to suggest actions to users that help improve their battery life. This can be done by building a battery usage model for each user.

***Data:*** The Carat app periodically collects a variety of data from the mobile phone it is running on, including the phone model, version of the operating system, the state of the battery, the CPU and memory utilization and the applications that are running. We use these fields to build a ML model that predicts the battery drain rate for a user. Using this model, it is possible to point out potential application that are responsible for an increased battery drain.

***Latency-Accuracy Trade-off:*** For users signing up for the Carat app, it is desirable to provide suggestions as soon as possible. However, currently, it takes several weeks for the app to collect enough data for a new user. Figure 5 shows the results of building a model for suggesting apps that are bugs for a particular user once enough data has been collected. It can be seen that a per-user model (denoted *Local*) works the best, but at the cost of latency. The local model performs poorly until enough data has been collected as depicted in fig. 6. A global model can be built immediately, but has poor accuracy. It is intuitive to think of grouping users who have the same model device together, or same operating system together. However, these grouping (denoted *Model* and OS) does not yield significant benefits. Further, as people install/uninstall apps, the models need to be updated. This make the domain ideal for testing CellScope's techniques.

***Extending Similarity Metric and MTL:.*** To extend our techniques to a new domain, we need to (*i*) customize the similarity metric (used for grouping) to the domain under consideration, and (*ii*) modify the MTL formulation in eq. (5) for this domain. In the cellular networks domain, our similarity metric was weighted by geographic distance between base stations. However, geographic distance does not have an effect here. From fig. 5, we notice that device model and operating system also do not make much difference either. Intuitively, the subset of apps common between the users should provide better results. However, just that alone is not enough as usage patterns vary across users with similar apps. The Carat dataset provides enough information to determine the number of times each app is active, which is roughly an indicator of the usage pattern for the user. We use that to derive usage similarity between users, $u_{usage_{(A,B)}}$, and utilize that to form the similarity metric:

$$SF_{CellScope} = u_{usage_{(A,B)}} \times \sum_{i=1}^{k} \sum_{j=1}^{n} |a_{ij} - b_{ij}|$$

The MTL formulation remains the same as in eq. (5), we simply replace $f_s$ with per-user features $f_u$.

We implemented a *Mobile Energy Diagnosis* module in CellScope at the same level as the RAN Performance Analyzer in fig. 3 that uses our modified similarity metric and MTL formulation. We then applied the grouping and learning to the measurement data we obtained to build a model

for suggesting bugs to a new user. The results are shown in fig. 6 which shows the accuracy of models built with (denoted *CellScope*) and without CellScope (denoted *per-user*) starting from the day a user installs Carat. We see that on the day of signing up, the accuracy of the model built without using CellScope is unusable. This is intuitive, since only a few samples have been sent by the new user's device. Over time, the user sends enough data and the accuracy improves. However, it takes over a week for Carat to offer usable suggestions to a new user. In contrast, with CellScope, we are able to build models that are immediately usable, and Carat can begin offering suggestions on day 1.

## 9 RELATED WORK

***Monitoring and Troubleshooting.*** Network monitoring and troubleshooting has been an active area of research in both wired networks [21, 28, 54] and wireless networks [3, 4, 13, 16]. These techniques do not employ machine learning for troubleshooting. Systems targeting RAN [4, 16] typically monitor aggregate KPIs and per-bearer records separately. Their root cause analysis of KPI problems correlates with aggregation air interface metrics such as SINR histograms and configuration data. Because these systems rely on traditional database technologies, it is hard for them to provide fine-grained prediction based on bearer models. Recent research [25] and commercial offerings [6] have looked at the problem of scalable cellular network analytics by leveraging big data frameworks. However, they do not support learning tasks. In contrast, CellScope focuses on scalable and accurate application of machine learning in such domains.

***Self-Organizing Networks (SON).*** The goal of SON [1] is to make the network capable of self-configuration (e.g. automatic neighbor list configuration) and self-optimization. CellScope's techniques can provide the necessary diagnostics capabilities for assisting SON.

***Modeling and Diagnosis Techniques.*** Problem diagnosis in cellular networks has been explored extensively in the literature in various forms [9, 23, 26, 34, 39, 49]. The focus of these has either been detecting faults or finding the root cause of failures. A vast majority of such techniques depend on aggregate information and correlation based fault detection. [26] discusses the shortcomings of using aggregate KPIs, and propose the use of fine-grained information. Some studies have focused on understanding the interaction of applications and cellular networks [24, 27, 38, 40, 52]. These are largely orthogonal to our work.

Finally, some recent proposals leverage the use of ML for specific tasks. In [49], the authors discuss the use of ML tools in predicting impending call drops and its duration. A probabilistic system for auto-diagnosing faults in RAN is presented in [9]. It uses KPIs as inputs to the model. [8] shows that improving signal-to-noise ratio, decreasing load and reducing handovers in cellular networks can improve web quality of experience by using ML to model the influence of radio network characteristics on user experience metrics. Our previous work [26] proposed the use of simple, explainable ML models towards the quest of automating RAN problem detection and diagnosis, and discussed several challenges in leveraging ML. In this paper, we present techniques that can solve the challenges in leveraging ML in many domains.

***Multi-Task Learning.*** MTL builds on the idea that related tasks can learn from each other to achieve better statistical efficiency [10, 11, 17, 50]. Since the assumption of task relatedness do not hold in many scenarios, techniques to automatically cluster tasks have been explored in the past [20, 30]. However, these techniques consider tasks as black boxes and hence cannot leverage domain specific structure. CellScope proposes a hybrid offline-online MTL formulation on domain-specific grouping of tasks based on the underlying performance characteristics.

## 10 CONCLUSION

The practicality of real-time mobile data analytics in many domains is impeded by a fundamental trade-off between data collection latency and analysis accuracy. In this paper, we first exposed this trade-off using the domain of cellular networks RAN. We presented CellScope to resolve this trade-off by applying a domain specific formulation of MTL. To apply MTL effectively, CellScope proposed a novel PCA inspired similarity metric that groups data from geographically nearby base stations sharing performance commonalities. Finally, it also incorporates a hybrid online-offline model for efficient model updates. Our evaluations show significant benefits. We have also used CellScope to analyze a live LTE network, where it could offer significant reduction in troubleshooting efforts. We then explored the generality of our techniques by applying them to a new domain, energy anomaly diagnosis in smartphones. We show that extending our grouping and learning techniques to a new domain is easy and effective. Thus we believe our proposals form a solid framework for mitigating the effects of latency-accuracy trade-off in real-time mobile data analytics systems.

## ACKNOWLEDGMENTS

We sincerely thank all Mobicom reviewers and our shepherd for their valuable feedback. In addition to NSF CISE Expeditions Award CCF-1730628, this research is supported by gifts from Alibaba, Amazon Web Services, Ant Financial, Arm, CapitalOne, Ericsson, Facebook, Google, Huawei, Intel, Microsoft, Scotiabank, Splunk and VMware. Mosharaf Chowdhury is supported by NSF grant CNS-1563095.

## REFERENCES


[1] 3gpp. [n. d.]. Self-Organizing Networks SON Policy Network Resource Model (NRM) Integration Reference Point (IRP). http://www.3gpp.org/ftp/Specs/archive/32_series/32.521/.

[2] Bhavish Aggarwal, Ranjita Bhagwan, Tathagata Das, Siddharth Eswaran, Venkata N. Padmanabhan, and Geoffrey M. Voelker. 2009. NetPrints: diagnosing home network misconfigurations using shared knowledge. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation (NSDI'09)*. USENIX Association, Berkeley, CA, USA, 349–364. http://dl.acm.org/citation.cfm?id=1558977.1559001

[3] Alcatel Lucent. 2013. 9900 Wireless Network Guardian. http://www.alcatel-lucent.com/products/9900-wireless-network-guardian.

[4] Alcatel Lucent. 2014. 9959 Network Performance Optimizer. http://www.alcatel-lucent.com/products/9959-network-performance-optimizer.

[5] Alcatel Lucent. 2014. Alcatel-Lucent Motive Big Network Analytics for service creation. http://resources.alcatel-lucent.com/?cid=170795.

[6] Alcatel Lucent. 2014. Motive Big Network Analytics. http://www.alcatel-lucent.com/solutions/motive-big-network-analytics.

[7] Paramvir Bahl, Ranveer Chandra, Albert Greenberg, Srikanth Kandula, David A. Maltz, and Ming Zhang. 2007. Towards Highly Reliable Enterprise Network Services via Inference of Multi-level Dependencies. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '07)*. ACM, New York, NY, USA, 13–24. https://doi.org/10.1145/1282380.1282383

[8] Athula Balachandran, Vaneet Aggarwal, Emir Halepovic, Jeffrey Pang, Srinivasan Seshan, Shobha Venkataraman, and He Yan. 2014. Modeling Web Quality-of-experience on Cellular Networks. In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking (MobiCom '14)*. ACM, New York, NY, USA, 213–224. https://doi.org/10.1145/2639108.2639137

[9] Raquel Barco, Volker Wille, Luis Díez, and Matías Toril. 2010. Learning of Model Parameters for Fault Diagnosis in Wireless Networks. *Wirel. Netw.* 16, 1 (Jan. 2010), 255–271. https://doi.org/10.1007/s11276-008-0128-z

[10] Jonathan Baxter. 2000. A Model of Inductive Bias Learning. *J. Artif. Int. Res.* 12, 1 (March 2000), 149–198. http://dl.acm.org/citation.cfm?id=1622248.1622254

[11] Richard Caruana. 1993. Multitask Learning: A Knowledge-Based Source of Inductive Bias. In *Proceedings of the Tenth International Conference on Machine Learning*. Morgan Kaufmann, 41–48.

[12] Ira Cohen, Moises Goldszmidt, Terence Kelly, Julie Symons, and Jeffrey S. Chase. 2004. Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6 (OSDI'04)*. USENIX Association, Berkeley, CA, USA, 16–16. http://dl.acm.org/citation.cfm?id=1251254.1251270

[13] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. 2003. Gigascope: a stream database for network applications. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data (SIGMOD '03)*. ACM, New York, NY, USA, 647–651. https://doi.org/10.1145/872757.872838

[14] Thomas G Dietterich. 2000. Ensemble methods in machine learning. In *Multiple classifier systems*. Springer, 1–15.

[15] Ericsson. 2012. Ericsson RAN Analyzer Overview. http://www.optxview.com/Optimi_Ericsson/RANAnalyser.pdf.

[16] Ericsson. 2014. Ericsson RAN Analyzer. http://www.ericsson.com/ourportfolio/products/ran-analyzer.

[17] Theodoros Evgeniou and Massimiliano Pontil. 2004. Regularized Multi–task Learning. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '04)*. ACM, New York, NY, USA, 109–117. https://doi.org/10.1145/1014052.1014067

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

[19] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A Survey on Concept Drift Adaptation. *ACM Comput. Surv.* 46, 4, Article 44 (March 2014), 37 pages. https://doi.org/10.1145/2523813

[20] Pinghua Gong, Jieping Ye, and Changshui Zhang. 2012. Robust Multi-task Feature Learning. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '12)*. ACM, New York, NY, USA, 895–903. https://doi.org/10.1145/2339530.2339672

[21] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. 2014. I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI'14)*. USENIX Association, Berkeley, CA, USA, 71–85. http://dl.acm.org/citation.cfm?id=2616448.2616456

[22] Paul Harris, Chris Brunsdon, and Martin Charlton. 2011. Geographically weighted principal components analysis. *International Journal of Geographical Information Science* 25, 10 (2011), 1717–1736.

[23] Chi-Yao Hong, Matthew Caesar, Nick Duffield, and Jia Wang. 2012. Tiresias: Online Anomaly Detection for Hierarchical Operational Network Data. In *Proceedings of the 2012 IEEE 32Nd International Conference on Distributed Computing Systems (ICDCS '12)*. IEEE Computer Society, Washington, DC, USA, 173–182. https://doi.org/10.1109/ICDCS.2012.30

[24] Junxian Huang, Feng Qian, Yihua Guo, Yuanyuan Zhou, Qiang Xu, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. 2013. An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13)*. ACM, New York, NY, USA, 363–374. https://doi.org/10.1145/2486001.2486006

[25] Anand Iyer, Li Erran Li, and Ion Stoica. 2015. CellIQ : Real-Time Cellular Network Analytics at Scale. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 309–322. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/iyer

[26] Anand Padmanabha Iyer, Li Erran Li, and Ion Stoica. 2017. Automating Diagnosis of Cellular Radio Access Network Problems. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking (MobiCom '17)*. ACM, New York, NY, USA, 79–87. https://doi.org/10.1145/3117811.3117813

[27] Haiqing Jiang, Yaogong Wang, Kyunghan Lee, and Injong Rhee. 2012. Tackling Bufferbloat in 3G/4G Networks. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference (IMC '12)*. ACM, New York, NY, USA, 329–342. https://doi.org/10.1145/2398776.2398810

[28] Srikanth Kandula, Ratul Mahajan, Patrick Verkaik, Sharad Agarwal, Jitendra Padhye, and Paramvir Bahl. 2009. Detailed diagnosis in enterprise networks. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication (SIGCOMM '09)*. ACM, New York, NY, USA, 243–254. https://doi.org/10.1145/1592568.1592597

[29] Gunjan Khanna, Mike Yu Cheng, Padma Varadharajan, Saurabh Bagchi, Miguel P. Correia, and Paulo J. Veríssimo. 2007. Automated Rule-Based Diagnosis Through a Distributed Monitor System. *IEEE Trans. Dependable Secur. Comput.* 4, 4 (Oct. 2007), 266–279. https://doi.org/10.1109/TDSC.2007.70211

[30] Seyoung Kim and Eric P. Xing. 2010. Tree-Guided Group Lasso for Multi-Task Regression with Structured Sparsity. *Intenational Conference on Machine Learning (ICML)* (2010).

[31] Tim Kraska, Ameet Talwalkar, John C. Duchi, Rean Griffith, Michael J. Franklin, and Michael I. Jordan. 2013. MLbase: A Distributed Machine-learning System. In *CIDR*. http://www.cidrdb.org/cidr2013/Papers/CIDR13_Paper118.pdf

[32] WJ Krzanowski. 1979. Between-groups comparison of principal components. *J. Amer. Statist. Assoc.* 74, 367 (1979), 703–707.

[33] Anukool Lakhina, Mark Crovella, and Christophe Diot. 2004. Diagnosing Network-wide Traffic Anomalies. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '04)*. ACM, New York, NY, USA, 219–230. https://doi.org/10.1145/1015467.1015492

[34] Yan Liu, Jing Zhang, M. Jiang, D. Raymer, and J. Strassner. 2008. A model-based approach to adding autonomic capabilities to network fault management system. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*. 859–862. https://doi.org/10.1109/NOMS.2008.4575232

[35] Adam J. Oliner, Anand P. Iyer, Ion Stoica, Eemil Lagerspetz, and Sasu Tarkoma. 2013. Carat: Collaborative Energy Diagnosis for Mobile Devices. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys '13)*. ACM, New York, NY, USA, Article 10, 14 pages. https://doi.org/10.1145/2517351.2517354

[36] Yan Pan, Rongkai Xia, Jian Yin, and Ning Liu. 2015. A Divide-and-Conquer Method for Scalable Robust Multitask Learning. *Neural Networks and Learning Systems, IEEE Transactions on* 26, 12 (Dec 2015), 3163–3175. https://doi.org/10.1109/TNNLS.2015.2406759

[37] K. Pearson. 1901. On lines and planes of closest fit to systems of points in space. *Philos. Mag.* 2, 6 (1901), 559–572.

[38] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Zhuoqing Mao, Subhabrata Sen, and Oliver Spatscheck. 2011. Profiling Resource Usage for Mobile Applications: A Cross-layer Approach. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys '11)*. ACM, New York, NY, USA, 321–334. https://doi.org/10.1145/1999995.2000026

[39] Sudarshan Rao. 2006. Operational Fault Detection in Cellular Wireless Base-stations. *IEEE Trans. on Netw. and Serv. Manag.* 3, 2 (April 2006), 1–11. https://doi.org/10.1109/TNSM.2006.4798311

[40] Sanae Rosen, Haokun Luo, Qi Alfred Chen, Z. Morley Mao, Jie Hui, Aaron Drake, and Kevin Lau. 2014. Discovering Fine-grained RRC State Dynamics and Performance Impacts in Cellular Networks. In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking (MobiCom '14)*. ACM, New York, NY, USA, 177–188. https://doi.org/10.1145/2639108.2639115

[41] Ahmed M Safwat and Hussein Mouftah. 2005. 4G network technologies for mobile telecommunications. *Network, IEEE* 19, 5 (2005), 3–4.

[42] Stefania Sesia, Issam Toufik, and Matthew Baker. 2009. *LTE: the UMTS long term evolution*. Wiley Online Library.

[43] Muhammad Zubair Shafiq, Jeffrey Erman, Lusheng Ji, Alex X. Liu, Jeffrey Pang, and Jia Wang. 2014. Understanding the Impact of Network Dynamics on Mobile Video User Engagement. In *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '14)*. ACM, New York, NY, USA, 367–379. https://doi.org/10.1145/2591971.2591975

[44] Muhammad Zubair Shafiq, Lusheng Ji, Alex X. Liu, Jeffrey Pang, Shobha Venkataraman, and Jia Wang. 2013. A First Look at Cellular Network Performance During Crowded Events. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '13)*. ACM, New York, NY, USA, 17–28. https://doi.org/10.1145/2465529.2465754

[45] Shai Shalev-Shwartz and Ambuj Tewari. 2011. Stochastic Methods for L1-regularized Loss Minimization. *J. Mach. Learn. Res.* 12 (July 2011), 1865–1892. http://dl.acm.org/citation.cfm?id=1953048.2021059

[46] Clint Smith. 2006. *3G wireless networks*. McGraw-Hill, Inc.

[47] Evan R. Sparks, Ameet Talwalkar, Virginia Smith, Jey Kottalam, Xinghao Pan, Joseph E. Gonzalez, Michael J. Franklin, Michael I. Jordan, and Tim Kraska. 2013. MLI: An API for Distributed Machine Learning. In *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013*, Hui Xiong, George Karypis, Bhavani M. Thuraisingham, Diane J. Cook, and Xindong Wu (Eds.). IEEE Computer Society, 1187–1192. https://doi.org/10.1109/ICDM.2013.158

[48] Technical Specification Group. [n. d.]. 3GPP Specifications. http://www.3gpp.org/specifications.

[49] Nawanol Theera-Ampornpunt, Saurabh Bagchi, Kaustubh R. Joshi, and Rajesh K. Panta. 2013. Using Big Data for More Dependability: A Cellular Network Tale. In *Proceedings of the 9th Workshop on Hot Topics in Dependable Systems (HotDep '13)*. ACM, New York, NY, USA, Article 2, 5 pages. https://doi.org/10.1145/2524224.2524227

[50] Sebastian Thrun. 1996. Is Learning The n-th Thing Any Easier Than Learning The First?. In *Advances in Neural Information Processing Systems*. The MIT Press, 640–646.

[51] Robert Tibshirani. 1994. Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society, Series B* 58 (1994), 267–288.

[52] Guan-Hua Tu, Yuanjie Li, Chunyi Peng, Chi-Yu Li, Hongyi Wang, and Songwu Lu. 2014. Control-plane Protocol Interactions in Cellular Networks. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*. ACM, New York, NY, USA, 223–234. https://doi.org/10.1145/2619239.2626302

[53] Helen J. Wang, John C. Platt, Yu Chen, Ruyun Zhang, and Yi-Min Wang. 2004. Automatic Misconfiguration Troubleshooting with Peerpressure. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6 (OSDI'04)*. USENIX Association, Berkeley, CA, USA, 17–17. http://dl.acm.org/citation.cfm?id=1251254.1251271

[54] He Yan, A. Flavel, Zihui Ge, A. Gerber, D. Massey, C. Papadopoulos, H. Shah, and J. Yates. 2012. Argus: End-to-end service anomaly detection and localization from an ISP's point of view. In *INFOCOM, 2012 Proceedings IEEE*. 2756–2760. https://doi.org/10.1109/INFCOM.2012.6195694

[55] Kiyoung Yang and Cyrus Shahabi. 2004. A PCA-based Similarity Measure for Multivariate Time Series. In *Proceedings of the 2nd ACM International Workshop on Multimedia Databases (MMDB '04)*. ACM, New York, NY, USA, 65–74. https://doi.org/10.1145/1032604.1032616

[56] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI'12)*. USENIX Association, Berkeley, CA, USA, 2–2. http://dl.acm.org/citation.cfm?id=2228298.2228301

[57] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized Streams: Fault-tolerant Streaming Computation at Scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13)*. ACM, New York, NY, USA, 423–438. https://doi.org/10.1145/2517349.2522737

[58] Ce Zhang, Arun Kumar, and Christopher Ré. 2014. Materialization Optimizations for Feature Selection Workloads. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14)*. ACM, New York, NY, USA, 265–276. https://doi.org/10.1145/2588555.2593678

[59] Alice X. Zheng, Jim Lloyd, and Eric Brewer. 2004. Failure Diagnosis Using Decision Trees. In *Proceedings of the First International Conference on Autonomic Computing (ICAC '04)*. IEEE Computer Society, Washington, DC, USA, 36–43. http://dl.acm.org/citation.cfm?id=1078026.1078407