APPLICATIONS OF MACHINE LEARNING FOR NETWORKING RESEARCH

by

JARED KNOFCZYNSKI

A THESIS

Presented to the Department of Computer and Information Science
and the Robert D. Clark Honors College
in partial fulfillment of the requirements for the degree of
Bachelor of Science

February 2022

An Abstract of the Thesis of

Jared Knofczynski for the degree of Bachelor of Science
in the Department of Computer and Information Science to be taken March 2022.

Title: Applications of Machine Learning for Networking Research

Approved: Dr. Ramakrishnan Durairajan
Primary Thesis Advisor

The application of machine learning (ML) to mitigate network-related problems continues to pose significant challenges for researchers and operators alike. For one, there is a general lack of labeled training data in networking communities, and labeling techniques popular in other domains are ill-suited due to the scarcity of operators' domain expertise. Additionally, networking issues are typically multi-tasked in nature, requiring the development of multiple ML models (one per task) and resulting in multiplicative increases in training times as the number of tasks increases. To address these challenges, we propose ARISE, a multi-task weak supervision framework for network measurements. ARISE uses weak supervision-based data programming to label network data at scale and applies multi-task learning (MTL) to facilitate information sharing between tasks as well as reduce overall training time. Using community datasets from the Center for Applied Internet Data Analysis (CAIDA), we show that ARISE can create MTL models that demonstrate improved classification accuracy and reduced training times when compared to multiple single-task learning (STL) models.

Acknowledgements

I would like to thank Doctors Ram Durairajan and Walter Willinger for their invaluable assistance in conducting my research and for serving on my Thesis Committee; this research would not have been possible without them. I would also like to thank Professor Casey Shoop, my mentor in the Clark Honors College who also served on my thesis committee, for his assistance in creating the final thesis document and for motivating me to push myself and the quality of my work beyond what had once been my wildest dreams. I am so grateful to have been surrounded by so many of my outstanding peers, each of whom has helped me to remain hopeful in the face of both a worldwide pandemic and the prospect of finishing my undergraduate degree. Lastly, I would like to thank my family for their unwavering support throughout my degree and for encouraging me to excel as we move through uncertain times into the uncertain future.

This work is supported by the National Science Foundation through grants CNS 1850297, OAC 2126281, and the University of Oregon Office of the Vice President for Research and Innovation (VPRI) Fellowship. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF or the University of Oregon.

Table of Contents

List of Figures

List of Tables

Glossary

Hyperparameters

High-level, abstract parameters used to define a machine learning model.

Latency measurement

A network measurement used to identify the speed of a network via the packets traveling across it.

Loss function

A reward metric used to guide a machine learning model towards learning correct classification decisions.

Round-trip time

The amount of time (measured in milliseconds) taken for a request to travel across a network from one host to a destination and back again.

Training, testing, and evaluation sets

Distinct, non-overlapping datasets used to train, test, evaluate, and refine a machine learning model.

Mathematical Notation

$\boldsymbol{N}$ denotes the number of items, entries, or values in a set.

$\boldsymbol{\sigma}$ denotes the standard deviation of entries or values in a set.

$\bar{\boldsymbol{x}}$ and $\boldsymbol{\mu}$ denote the average (arithmetic mean) of entries in a set $\boldsymbol{x}$.

$\boldsymbol{x_i}$ denotes the i^{th} value or entry in a set $\boldsymbol{x}$.

$\sum \boldsymbol{x}$ denotes the sum of all entries in a set $\boldsymbol{x}$.

$\vee$ denotes the logical `OR` operation between boolean values (True or False), which returns True if any of its inputs are also True.

$[\boldsymbol{a}, \boldsymbol{b}]$ denotes a range of values stretching from $\boldsymbol{a}$ to $\boldsymbol{b}$, inclusive.

CHAPTER 1: INTRODUCTION

The application of machine learning (ML) in the field of networking has received mixed reactions from the network research and operator communities. On one hand, there exists great interest and enthusiasm among network researchers in developing new machine learning models for an increasing number of diverse network management tasks, both performance- and security-related [1]. However, this enthusiasm has been hampered by a general lack of training data (especially *labeled* training data) and has generated little to no interest in demonstrating whether or not a trained ML model will generalize as expected in deployment scenarios (i.e., if it can be trusted in practice). At the same time, operators have been slow to jump on the 'ML bandwagon' and remain reluctant to deploy untested ML-based tools in their production networks, voicing an overall dissatisfaction with black-box models that cannot be trusted because they fail to provide insight into how they work or how their decision making compares to that of a domain expert (i.e., network operator) [1], [3]. Rather than deploy untested ML-based tools in their networks, operators must to rely on several methods of naïve statistical techniques in conjunction with manual oversight [4], resulting in increased workloads and the inability to scale network oversight.

In this work, we take a step towards bridging this gap between what network researchers perceive as top priorities as far as their ML-based efforts are concerned and what operators look for in terms of benefiting from recent innovations in ML and applying them in practice. In particular, we address some of the challenges posed by (i) the increasing number of network management tasks for which researchers seek to develop automated ML-based tools despite a general paucity of suitable (labeled) data, and (ii) a

general inability of ML models to benefit from common characteristics that may be present across features when training different task-specific models, and thus pose a crucial roadblock for any widespread adoption of ML-based network management tools in practice.

To better articulate these challenges, consider the following network management scenario that we will use throughout the paper as an illustrative example, where a network operator seeks to identify certain patterns or traffic behaviors occurring on their network. This scenario involves just two tasks:

- **Task 1:** *Remove noisy measurements from time-series latency data and identify changepoints occurring after a congestion event.*
- **Task 2:** *Remove noisy measurements from time-series latency data and detect changepoints leading to a loss of packets in the network.*

Note that both tasks consist of multiple sub-tasks (i.e., removing noise, identifying changepoints, and detecting congestion in the case of Task 1; and removing noise, identifying changepoints, and identifying loss in Task 2). Here, the occurrence of a sudden spike in latency measurements (referred to as *network volatility*) can be indicative of noise in the data. Similarly, sustained periods of network volatility can indicate a persistent congestion event. Thus, for both congestion and noise detection, network volatility plays a significant yet common role. However, it is unclear how one might best exploit such common features in training an ML model, and what the performance implications of incorporating or ignoring such commonalities may be. In addition, the effort necessary to train a model becomes multiplicative when considering multiple sub-

tasks, as each sub-task requires training its own ML model, a process then further complicated by the paucity of labeled data.

To address these challenges and in an effort to catalyze the research and operator communities' efforts on the application of ML for networking tasks, we report in this paper on the design and implementation of a novel machine learning framework named ARISE. At its core, ARISE combines the use of weak supervision-based data programming strategies to label vast quantities of network measurements via *labeling functions* (programmatic representations of domain knowledge and heuristics related to specific networking tasks) with multi-task learning (MTL) capabilities to share information across tasks during the training process. As a result, ARISE provides a practical framework for supporting the efficient training of multiple ML models in ways that exploit the commonality of distinct tasks without reducing accuracy and that affords opportunities for reasoning about inferences made by the trained ML models.

Recent studies that focus exclusively on traditional single-task learning (STL) techniques have begun to tackle some of the corresponding issues that naïve statistical methods and STL models face, including the use of campus networks as rich data sources to overcome the data problem that has plagued the application of ML in the networking domain [5], the application of scalable techniques to overcome the data labeling problem [2], and the development of customized models for individual network management tasks (see Chapter 2, Section C). But unlike these state-of-the-art efforts, ARISE goes beyond STL models and provides several significant novel benefits as follows.

- The resulting MTL models feature layers that are shared across multiple tasks, as well as individual task-specific head layers. Importantly, the shared layers are trained only

once, whereas new tasks can be added "on the fly" by training only the head layers, thus drastically reducing model training times.

- Using shared layers, it is possible to capture commonalities among features across different tasks in a principled manner. Multi-task learning has already proven to generalize better than STL models due to *implicit data augmentation* [6], the sharing of information across tasks in the model resulting in an enhanced availability of data on which to train.
- With MTL models, available domain knowledge can be succinctly captured in several ways, including through the use of labeling functions, in the design of shared layers, and by deciding which task-specific head layer to call by the shared layer. This feature of ARISE can be leveraged to begin reasoning about the decisions taken by MTL models at the sub-task level in a qualitative manner.

To evaluate ARISE, we compare the F1 scores and training times of MTL and STL models across different sub-tasks such as loss, congestion, noise, and changepoint classification using data gathered by the CAIDA Ark project [7]. We also compare the resulting F1 scores of both STL and MTL models to similar classification tasks conducted via naïve statistical techniques and demonstrate that naïve methods are extremely ineffective in the context of network measurements. Our results show that MTL models can be trained with over 40% accuracy improvements (as measured by the F1 score) and up to 8x faster when compared to STL models, and are especially performant in the context of larger and noisier datasets. We show that MTLs generalize better and exhibit reduced training overheads for new sub-tasks (thus facilitating model scaling) and can be used to perform classification tasks on measurement data while enhancing an operator's ability to reason about classification decisions that the model makes at the sub-task level.

CHAPTER 2: BACKGROUND AND MOTIVATION

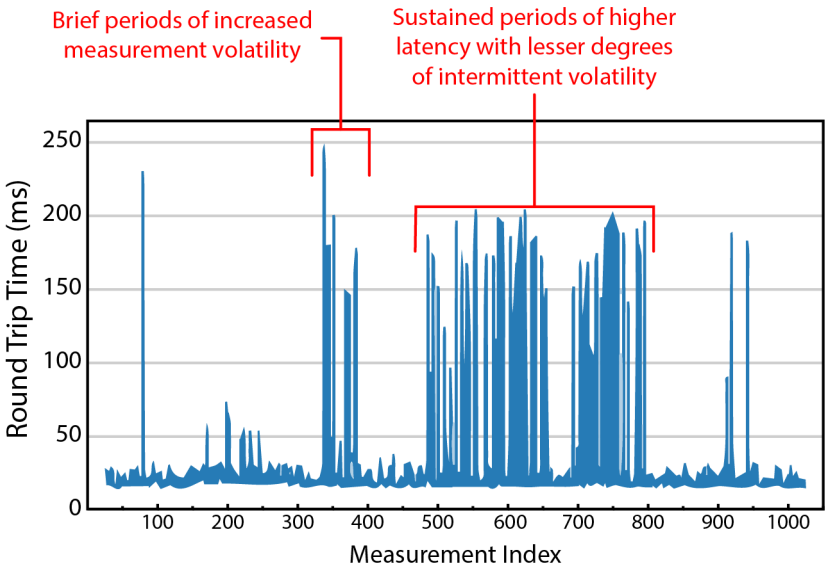
With growing traffic volumes and continued innovations in modern network systems, operators are struggling to keep pace with the rate of expansion and lack the capacity to effectively utilize all the measurement data they routinely collect to perform an ever-growing number of tasks (e.g., congestion detection, identifying noise, etc.). This has resulted in renewed interest in the efficacy of ML-based tools to assist in processing the available measurement data while simultaneously automating the execution of a myriad of tasks.

A. MOTIVATION

1) NETWORKING PROBLEMS INVOLVE TASKS COMPOSED OF MULTIPLE SUB-TASKS WITH OVERLAPPING CHARACTERISTICS

Many network problems exhibit telltale indicators of their presence in measured network data (e.g., latency measurements, NetFlow, etc.). Features such as large proportions of measurement noise or high levels of network congestion often act as symptoms of other underlying issues (e.g., packet loss events or denial of service attacks). Hence, detecting network-related problems with ML-based approaches typically involves the execution of classification tasks that are multi-task in nature. For example, consider the two sample tasks, **Task 1** and **Task 2**, introduced in Chapter 1. Each of these tasks are of potential interest to network operators and researchers alike, and both tasks are inherently dependent on multiple sub-tasks. For example, **Task 1** involves identifying and removing noise present in the input data, identifying network congestion events, and detecting subsequent changepoints in the latency measurements. **Task 2** involves noise removal and changepoint classification, followed by loss detection.

The aforementioned characteristics can take many different forms within a network, and there is often a significant degree of overlap between these characteristics across sub-tasks. We refer to these overlapping features as *composite characteristics,* where some aspect of the data is shared between the features being identified. For example, consider noise and congestion events as they occur on a network. Figure 1 depicts a brief period of latency measurement data (obtained from CAIDA's Ark project [7]; more details in Chapter 4, Section A) across the measurement indices of time-series latency data akin to that which a network operator or researcher may observe. Within the depicted period, we hypothesize two distinct network events (noise and congestion) that an operator or researcher may wish to detect.



On the left of Figure 1 (from measurement index 300 to 400), we see a brief moment of *network volatility*, which we define as a latency spike in the time series data that significantly exceeds the average round-trip times (RTTs) of prior measurements. An operator may wish to classify this temporary behavior as *measurement noise* to either

remove it from consideration for the task at hand (e.g., when seeking to analyze the overall behavior of the network without the presence of confounding variability in latency measurements), or to employ it as an indicator of some other behavior that may be of interest. Shortly after this stint of network volatility, a similar pattern appears when the latency measurements spike and remain increasingly volatile for a more sustained period of time (see measurements between indices 500 and 800 in Figure 1). This behavior may indicate a period of *network congestion*, where a sustained increase in traffic rates may result in additional latency in communications between network hosts. We do not attempt to rigorously define noise and congestion, and instead use these as examples of characteristics a network operator may wish to classify (using their heuristics/domain expertise) for any number of reasons. Despite their semantic independence from one another, these instances of noise and congestion exhibit similar composite characteristics in the underlying patterns of the data. That is, both events are represented by a sustained increase in overall latency, as well as in the variation between successive latency measurements, and a network operator may wish to identify the root cause(s) of this observed behavior.

Given the prevalence of noise and congestion in network-related issues as well as the degrees of similarity between their underlying characteristics, we use noise and congestion as two canonical examples of sub-tasks in this paper. We also employ loss (outage) and changepoint classification as two additional examples of sub-tasks that could be useful in identifying and addressing more significant network problems. The classification of each of these features involves (in one way or another) the variation of successive latency measurements in network data. For example, a changepoint may occur

when the consistent behavior of a network shifts from one behavioral pattern to another (such as before, during, or after a congestion event). Similarly, a packet loss event may occur whenever the connection fails or the measured latency drops to zero.[1]

2) MODEL SCALING IS IMPOSSIBLE WITH SINGLE-TASK LEARNING

Despite the presence of composite characteristics in the aforementioned sub-tasks, traditional ML-based approaches are limited in their ability to adapt this information to the training of different task-specific models. For example, to form a cohesive network classification task, traditional ML techniques rely on several distinct learning models (one per sub-task) where each model is trained separately, then chained together to obtain holistic, task-specific answers. We refer to the models based on this stepwise approach as single-task learning models, or STLs.

Training an STL for a new sub-task "on the fly" (henceforth referred to as *model scaling*) will result in a multiplicative increase in training times. This is due to the fact that the new sub-task must be trained, tweaked, tuned, and re-trained separately from the remainder of the model. Even in cases where additional compute resources are readily available (e.g., via cloud offloading of ML model training) or where parallel training of multiple STL models (each per sub-task) is feasible, the total training time to train all STLs for a given task is equal to the longest training time among the individual STL models.

[1] In our datasets, packets that failed to arrive at their destination were recorded with a latency of 0, indicating packet loss.

Beyond simply increasing the amount of training time required for holistic model development, the lack of information sharing in STL models—especially in the face of composite characteristics—also has a negative effect on the classification accuracy of each individual model. Because of this lack of information sharing, each individual model effectively discards any information it gleans during its training process that it deems as irrelevant for the task at hand, even if that information may be relevant in the context of a different task. In effect, the model is limited in its degree of accuracy and overall generalizability as a direct result of its lack of information sharing.

Furthermore, the black-box nature of commonly-used STL techniques makes it difficult for network operators to gain insight into the root cause of individual classification decisions (e.g., [8]). In other words, when a given model returns a specific decision, it is generally impossible for the network operator to reverse engineer the model's decision-making process to better understand why and how the model arrived at its decision and not at some other outcome.

B. CHALLENGES

In light of these pitfalls of STLs, we argue that to be of practical interest to operators and to encourage researchers to tailor ML to the needs of networking, an "ideal" ML-based approach to inferring network problems from measurements should address the following challenges [1], [3]:

- **Efficient training of ML models.** The addition of a new network task should not require retraining the model from scratch; that is, scaling a model to incorporate new tasks should incur minimal overhead with respect to training time. This issue is further

complicated by the fact that the networking domain largely lacks access to readily available and high-quality labeled datasets.

- **Maintaining model accuracy.** Training models for network-related tasks should account for and exploit the presence of composite characteristics across sub-tasks; that is, information sharing among different models should be a pre-requisite. However, exploiting composite characteristics across sub-tasks may also result in model underfitting, as the additional constraints imposed by the presence of different features may negatively impact the model's overall accuracy.

C. Prior Efforts and their Limitations

Applying ML to solve networking problems is of great importance to the network research and operator communities, and previous efforts in this area can be roughly grouped into three categories. The first category concerns techniques based on supervised learning (e.g., regression) to infer a continuous variable (e.g., RTT, loss), or on classification to assign observations to a set of pre-established classes (e.g., traffic types). For surveys of supervised learning-based techniques, see [9] and [10]. Complementary to these techniques are statistical methods such as expectation maximization [11], [12], which allow a model to adjust based on the statistical properties of the data's distribution. Two key shortcomings of these techniques are (i) their inability to accurately capture operators' domain knowledge, and (ii) their inherent need for labeled datasets that are hard to come by in the networking domain.

One example of such statistical methods is the naive classification technique described in [4] and [2], which evaluate the efficacy of using simple filters such as $\mu + 3\sigma$ to classify latency measurements as noise. Yet as these prior efforts show, the isolated

use of naive techniques is insufficient to accurately classify instances of noise in measurement data; to attain sufficient accuracy, the use of ML is essential.

However, noise is not the only network characteristic that operators may wish to classify. They may also wish to identify other features such as *changepoints* (points at which the underlying behavior of a time series changes), for which there already exist several naive classification solutions an operator may attempt to employ. Some state-of-the-art techniques for changepoint classification (e.g. [13]) allow operators to identify changepoints in recorded data, but many of these tools still struggle to accurately quantify changepoints in network data without significant operator input. For example, *ruptures* [14], a changepoint identification framework, offers a suite of naive changepoint classification methods including PELT [15], kernel-based changepoint detection [16] binary segmentation [17], [18], bottom-up segmentation [19], [20], and more, each of which offer a collection of benefits and significant drawbacks in the context of networking applications. Specifically, these naive techniques rely upon additional pieces of contextual information to operate effectively which network operators cannot reasonably be expected to provide. For one, several of these classification methods require the expected number of changes in a given time-series to be known a priori, which a network operator cannot possibly know or accurately predict. Additionally, many of these methods require further operator input in the form of cost and smoothing functions to apply to the datasets and generate thresholds that can be used to identify changepoints in a time-series. These functions require an inherent understand of the distribution of data which operators largely do not possess, particularly in live or real-time management scenarios. These methods are also easily influenced by measurement noise, making it

difficult to employ them effectively without a considerable amount of manual tuning to identify the proper thresholds. This is extremely impractical for network operators, as with so few resources, the effort expended to tweak and tune these naive changepoint classification methods may be better spent manually identifying changepoints in relevant measurement datasets. These methods also require that the entire collection of time-series data being analyzed be gathered and known beforehand, and as such, could not be used effectively in online, real-time network management scenarios. To confirm that such naïve efforts are indeed ineffective in the context of networking, we briefly evaluate *ruptures*' efficacy as a changepoint identification tool in the context of Internet measurements.

The *ruptures* [14] Python library for offline changepoint detection provides a function to generate time-series data by sampling from either a uniform or Gaussian normal distribution and prescribing specific changepoints throughout the time-series with which to compare. To evaluate its performance, we sample 1,000 baseline values from a uniform distribution containing three changepoints at indices 250, 500, and 750 of the time-series. We then employ the binary segmentation [18] classification method with *ruptures*' default recommended parameters to observe its effectiveness. Note that in this case, the binary segmentation method is provided the *number of changepoints* to expect and classify, which would be impossible for an operator to predict in any applied context.

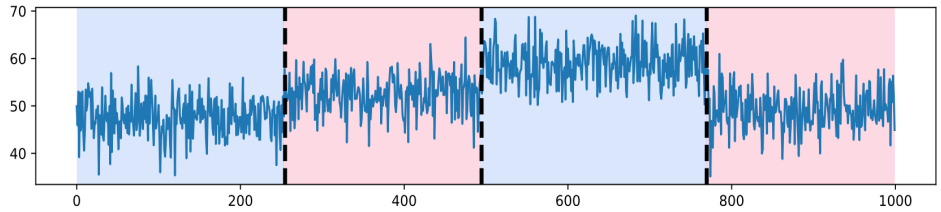


Figure 2: Using *ruptures* to classify changepoints. True changepoints are shown by the transitions in background color, predicted changepoints are shown via the dotted lines.

As Figure 2 shows, *ruptures* is quite capable of identifying static changepoints when provided the number of changepoints to expect and in the presence of relatively little noise in the data. However, when attempting to classify more complex examples where the number of changepoints are not known or only a portion of the time-series is available on which to classify (as is often the case when working with networking or measurement data), *ruptures'* classification capabilities quickly fall short.

We again sample 1,000 points from a uniform distribution, this time with 10 prescribed changepoints contained within the data. We use the window-sliding segmentation method of changepoint classification to identify changepoints *without providing the number of changes to expect*, and instead provide the method with a penalty function $p = \ln(N) \cdot \sigma^2$ (*ruptures'* recommended default, where N is the number of samples and σ is the standard deviation of the noise) to isolate changepoints without needing to know the number of changes beforehand.

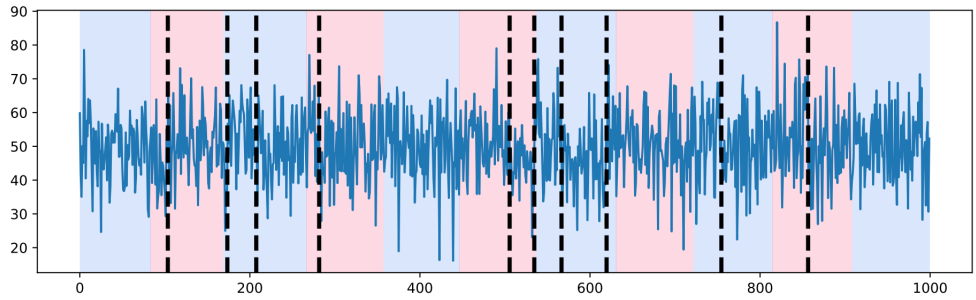


Figure 3: Classifying an unknown quantity of changepoints using window-sliding segmentation. True changepoints are shown by the transitions in background color, predicted changepoints are shown via the dotted lines.

As Figure 3 suggests, the window-segmentation method struggles to accurately classify changepoints on time-series data when not provided the number of changes to expect. And while the penalty function could be modified further to better represent the distribution of this data, such efforts would be impractical (if not impossible) for a network operator to perform in the context of each of their individual datasets. Other changepoint classification efforts similar to *ruptures* suffer from similar limitations in that they

(i) Require additional contextual information regarding the distribution of time-series data that network operators cannot possess, and

(ii) Are extremely vulnerable to noise and suffer when not provided sufficient information regarding the behavior of the data.

Thus, we turn to ML to address the shortcomings of these naive techniques.

The second category concerns efforts that focus on overcoming the need for domain expertise and large, labeled datasets, consisting primarily of unsupervised learning techniques such as clustering and Principal Component Analysis (PCA) to detect anomalies in network data (e.g., BGP [8] or traffic measurements [21]–[24]), to help with network diagnosis [25], and to perform event detection [26]. These efforts suffer from a

general inability to interpret their decisions or reason about their underlying decision-making processes, and therefore pose a significant obstacle with respect to the adoption of ML-based methods to address networking problems. For example, Ringberg et al. [21] show that PCA-based techniques for network classification problems are sensitive to changes in the level of traffic aggregation, and their efficacy may be polluted by large anomalies in the available network data. These factors make it difficult for network operators to successfully employ PCA-based techniques due to the extent of manual tuning required to achieve sufficient performance and due to their lack of generalizability to diverse data sources or types. These observations point towards an urgent need for models and techniques that provide operators with opportunities to reason about the inferences made and to develop trust in the inference processes.

The third category of efforts consists of learning techniques that leverage weakly supervised learning. These techniques attempt to combine and learn noisy labels from many weak sources to build a predictive model. The most popular forms of weak supervision are distant supervision [27], [28] and crowdsourcing [29], [30], both of which suffer from issues such as inaccuracy and inadequate coverage. Addressing these issues is the dual objective of Snorkel [31], which combines labels from different weak supervision sources to increase the accuracy and coverage of training sets using *data programming*, where users can programmatically create lower-quality training datasets [32]. Inspired by this data programming paradigm, two recent efforts [2], [4] seek to classify network features using weak supervised learning. However, these efforts are not yet conclusive on this topic. For one, while effective in classifying network noise, both efforts are not designed to scale with the addition of new sub-tasks or the performance

demands imposed by larger training datasets. Furthermore, while sufficient for the classification of a single sub-task, both approaches are lacking with respect to their abilities to handle multiple tasks and leverage the composite characteristics of distinct features across sub-tasks. Lastly, neither of these efforts provide any means to begin reasoning about decisions taken by the models at local (e.g., LIME [33], SHAP [34]) or global (e.g., Bastani et al. [35]) levels.

CHAPTER 3: DESIGN AND IMPLEMENTATION OF ARISE

In this section we provide an overview of ARISE, describe the design details of its individual components, and conclude by providing the technical specifications of our implementation.

A. OVERVIEW OF ARISE

The main goal of this work is to design and build ARISE, a novel machine learning framework motivated by the demands of networking as an application domain for ML that addresses the aforementioned challenges. ARISE combines two key ideas, namely, (a) the use of weak supervision-based data programming strategies to label vast quantities of network measurements via labeling functions, and (b) the use of multi-task learning (MTL) to enable the sharing of information across sub-tasks during the training process. While each of these ideas has been considered individually and evaluated extensively in the ML community, to the best of our knowledge, ours is the first effort to marry their strengths to address limitations of ML in networking.

Key Insights: At a high-level, the augmentation of weak supervision with multi-task learning provides the following key benefits, each of which will be described in detail in Section B below. For one, the MTL models that ARISE creates feature a set of layers that are shared across multiple sub-tasks (i.e., "shared" layers) and a set of task-specific individual layers (i.e., "head" layers). These shared layers are trained only once while new sub-tasks can be added "on the fly" by training only the lightweight head layers. This division of labor facilitates model scaling, drastically reducing model training times and addressing the first design challenge. Furthermore, by leveraging the shared layers, researchers and operators can capture composite characteristics across different tasks

(e.g., the common role played by the network volatility feature described above). ARISE relies on a multi-task classification technique (described below) that allows the model to generalize better due to *implicit data augmentation*. This technique ensures the accuracy of the model and addresses the second design challenge.

B. Design Details

Figure 4 depicts the overall architecture of the ARISE framework. We describe each of the components below.

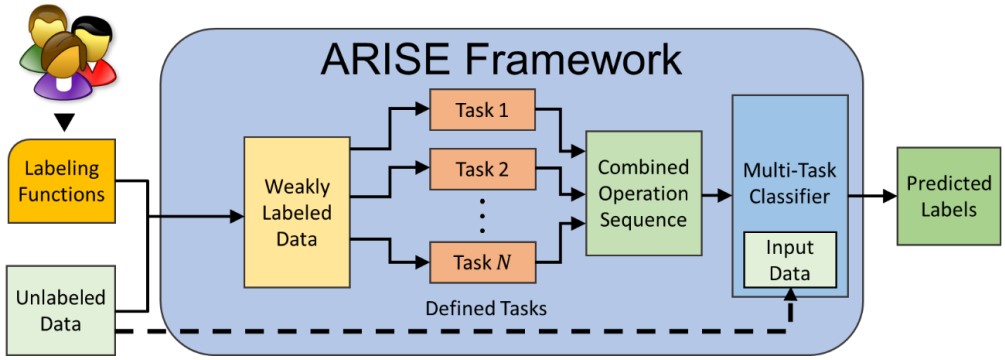


Figure 4: Design of the ARISE framework.

User Interface. The success of ARISE hinges on its ability to capture the knowledge and domain expertise of network researchers and operators, and convert such information into concrete, tangible heuristics expressed as literal lines of code (e.g., if $RTT_i > \mu + 3\sigma$, then RTT_i is a noisy measurement in the dataset). To this end, we have designed a simple interface that operators and researchers can use to convert their domain knowledge about network events into programmatic representations (e.g., using Python) referred to as labeling functions.

Unlabeled Data. Datasets are collected using traditional measurement techniques (e.g., using `scamper` [36]) or based on vendor-specific representations (e.g., NetFlow). More

details about the datasets used in this study are described in Chapter 4, Section A. The only assumption that we make in the design of ARISE is the possibility of extracting time series measurements from a dataset. Time series are a commonly used method for operators to obtain a temporal view of their data so they can assess networking problems (e.g., to check the effectiveness of a mitigation solution by looking at traffic patterns before and after a congestion event).

```
def label_noise(latency, alpha):
    noise_threshold = 1.5 * alpha
    if latency >= noise_threshold:
        return VOTE
    else:
        return NORMAL

def label_congestion(lat, alpha, beta):
    con_max = 1.5 * alpha
    con_min = 1.2 * beta
    if lat in range(con_min, con_max):
        return VOTE
    else:
        return NORMAL
```

Labeling Functions. To generate labels for network characteristics, operators can construct simple labeling functions that can then be applied to the unlabeled input data, borrowing the idea of data programming from Ratner et al. [32]. To illustrate, we use *tsfresh* [37] to extract feature-specific threshold values for each dataset and show how to use those thresholds to label data points with the help of the noise labeling function `label_noise` shown below (`NORMAL` and `VOTE` correspond to the integer values 0 and 1, respectively). This function labels each point as noise or normal while still accounting for the baseline differences in each dataset's individual behavior. More concretely, to label a point as `NOISE`, the function first extracts the threshold value $1.5 \cdot \alpha$, where α is

the value of the RTT at the 75th percentile for the current dataset. It then returns `VOTE` (a positive label) if the latency value for a given measurement exceeds the specified threshold.

A similar function can be used to label a point as `CONGESTION` if the RTT of the input measurements falls within the range $[1.2 \cdot \beta, 1.5 \cdot \alpha]$, where β is the RTT value at the 30th percentile for the current dataset. We can also construct a `LOSS` labeling function that returns a positive label if the input RTT is zero, as in our datasets, a measurement of zero indicates that a packet was lost. These thresholds were selected by manually examining several of the input datasets to determine an appropriate latency range for each feature being classified. Other users of ARISE may easily implement their own thresholds and labeling functions by examining their data and translating their domain heuristics into concrete labeling functions such as the `label_noise` and `label_congestion` functions shown above. While our labeling function examples are arguably simple, more complex labeling functions could be constructed to isolate and identify more nuanced relationships between patterns in the data. As labeling functions are intended to distinguish between characteristics based on an operator's approximations of the underlying data, they can (and should) be adapted for specific contexts based on an operator's understanding of their given dataset.

Weakly labeled data. Applying labeling functions to unlabeled datasets results in the generation of weak labels, which can then be used to train and evaluate a downstream classification model. To this end, users must separate the weakly labeled data into training, testing, and validation sets, just as they would do when developing an STL model. This ensures that each task in the MTL model has access to the proper "ground

truth" labels generated by the labeling functions that it will evaluate itself against to improve its performance during the training process. Once the data has been labeled and partitioned into these training and evaluation sets, we then construct the isolated classification layers that perform each individual sub-task.

Tasks. In ARISE, each task entails multiple sub-tasks, and each sub-task corresponds to a specific task head responsible for leveraging the insights of the shared layer to perform feature classification on the input data. We define one head per task or sub-task and store them as sequences of operations to be used in the next stage. For example, a model seeking to perform **Task 1** (defined in Chapter 1) would employ four distinct task heads – one for each of the sub-tasks (noise, changepoint, and congestion classification), and one for the composite task representing the union of all three. When defining task heads, we also assign and specify any relevant implementation parameters, such as the desired loss, activation, or output functions of the task head, as well as the scoring metrics (e.g., F1 score) used to evaluate each task's performance. Once the heads are defined, we merge them into a combined operation sequence that acts as a logical container for the individual classification elements in the MTL model.

Combined operation sequence. The individual task heads all rely on the same shared layer(s) to process the input data and generate insights regarding the underlying patterns that comprise the features being classified (e.g., detecting network volatility as described in Chapter 2, Section A-1). By combining the individual operation sequences associated with each sub-task, we can construct a shared layer that feeds into each of the respective task heads in the model. We can also adjust the design of the combined operation

sequence to modify the model's information sharing capabilities, allowing us to specify exactly what sources of information are shared between which layers of the model.

In our implementation of ARISE, the operation sequences are designed to facilitate complete information sharing between all tasks. By enabling this complete information sharing, the shared layer can learn both the individual and composite characteristics that comprise the features being classified. This also acts as a form of *implicit data augmentation*, where each task receives additional information to learn from during the training process — namely, the output of other tasks. This enables the MTL model to "reuse" information learned in the shared layer in the classification of multiple tasks and allows individual task heads to benefit from the presence of others by learning how other tasks' predictions correlate with their own. Implicit data augmentation also reduces the risk of underfitting a model to the tasks at hand, as while each task employs the shared layer in its classification decisions, the task heads operate independently from one another, allowing them to make their decisions irrespective of the restrictions posed by other task heads. By the same logic, it also reduces the risk of overfitting the model to any one specific task, as the MTL classifier must optimize its performance with respect to each of its classification tasks during training.

We note that some implementations of ARISE may wish to employ alternative information sharing paradigms other than complete connectivity. By specifying exactly which layers of the MTL model share information with one another, one may isolate specific relationships between network characteristics or remove the influence of two unrelated, dissimilar tasks from one another in the training process. Once the model

structure has been established, we pass the list of tasks and the operation sequence to the multi-task classifier for the training of the final model.

Multi-task classifier. After applying weak labels to the data, constructing individual task heads, and architecting the flow of the shared layer, all that remains is to develop and train the final model. After establishing the model's hyperparameters for a given iteration and passing in the operation sequence of tasks, the model can be trained and consequently evaluated on the testing sets of input data by comparing the model's predicted labels to the "ground truth" labels generated by the defined labeling functions. In this stage, feedback may also be introduced in that the user may wish to continue modifying and retraining the model's hyperparameters until an optimal configuration is achieved. By repeating each of the previous steps, the users may define new heuristics via the given labeling functions, or they may wish to simply adjust the information sharing capabilities of the model after its initial configuration, which can be done by changing the model control schemes during the architecting of the shared or head layers. Given the modular nature of ARISE, new tasks can easily be added or removed from the framework "on the fly," allowing for the quick adjustment of the underlying structure in a principled and efficient manner.

C. IMPLEMENTATION

To implement ARISE, we leverage weak supervision components from Muthukumar et al. [4] and multi-task learning components from Ratner et al. [38].[2] From each dataset, we extract a number of features and time series characteristics using *tsfresh*

[2] We thank the authors of these frameworks for open sourcing their code to the community.

[37] that we then employ as thresholds in the development of our labeling functions (described in Section B above). Next, the input data to the ARISE pipeline is initially processed by a multilayer perceptron (MLP) module, then sent to task-specific head modules that perform the actual classification. This MLP module enables each task to 'eavesdrop' on the others, allowing them to learn from labeling functions and heuristics they are not explicitly associated with. For example, the `NOISE` classification task likely identifies and categorizes measurements it observes that exhibit increased latency beyond the norm for a given dataset. After the shared layer learns to recognize this feature in the input data, when the `CONGESTION` classifier attempts to identify similar characteristics, it can leverage the same insights from the hidden layer, allowing it to exploit existing information already maintained within the model to train a more accurate classifier in a shorter period of time.

Our multi-task model utilizes a standard multi-task classifier [38] and employs two densely connected hidden layers shared between all tasks that subsequently feed into the N task-specific head layers. In comparison, the conventional STL models against which we evaluate ARISE employ two hidden layers per task. This means that for a model or application requiring N distinct sub-tasks, the single-task method would require the training of $2N$ hidden layers, while our multi-task approach would only require $2 + N$ total layers to be trained. This improvement can result in a drastic decrease in model training time as we show in Chapter 4, Section C.

We base the single-task component of our analysis on the framework described in [4], which is built on a previous version of Snorkel (v0.7.0b0) to perform weak supervision in the context of a single-task noise classification. We modified this

framework to include additional capabilities such as classifying congestion, outages, and changepoint detection by writing new labeling functions. All models were trained and evaluated on a CloudLab.US [39] server running Ubuntu 16.04 LTS on an Intel Xeon D-1548 64-bit processor with 8GB of memory.

CHAPTER 4: EVALUATION OF ARISE

In this section, we describe our datasets and the experiments we conduct to evaluate the efficacy of the ARISE framework. For more information regarding the selection and tuning of model hyperparameters, see Appendix .

A. DATASETS USED

In our experiments, we use `traceroute` datasets generated with the `scamper` tool [36] from the CAIDA Ark [7] project. From the CAIDA dataset, we select 28 distinct network source/destination pairs from the `atl2-us` vantage point located in Tucker, Georgia, with measurements spanning 24 hours taken on January 1, 2019, to serve as training sets containing a total of 75,359 latency measurements.[3] These subsets contain between 2,000 to 4,000 measurements each, with an average of nearly 2,700 round-trip time measurements per source/destination pair. We then average the training times and F1 scores of each model on each subset of the data in the evaluation below and compare the results to evaluate the efficacy of both our STL and MTL approaches.

B. EXPERIMENTS

To evaluate ARISE, we initially train three single-task generative models based on the EMERGE framework [2] to individually classify latency measurements in a network time series as either `LOSS`, `NOISE`, or `CONGESTION`. For our ground truth, we use the output of our original labeling functions under the assumption that operators can define their own LFs using their domain expertise to accurately represent the characteristics of

[3] The CAIDA UCSD IPv4 Routed /24 Topology Dataset - January 1, 2019, https://www.caida.org/catalog/datasets/ipv4_routed_24_topology_dataset.

the features they wish to classify. We also train and compare the efficacy of the MTL models with three naïve statistical classifiers i.e., Overly Robust Covariance Estimation or ORCE, Elastic Ellipse or EE, and $\mu + 2\sigma$. The naive classifiers, typically used by network operators [2], serve as baseline in the absence of ground truth data.

We record the times taken for each model to train, as well as the F1 scores[4] of each task upon completion. We use the F1 score as our primary evaluation metric as it allows us to more effectively account for the potential sparsity of some features amongst those we seek to classify, as some features we seek to identify may not exist in a given measurement flow at all. After establishing this baseline, we move to perform the same analyses via the multi-task approach, using the same labeling functions and evaluation metrics with ARISE, recording the model training times and F1 scores upon completion.

To demonstrate the scalability of ARISE, we also examine the effects of adding new classification tasks into the pipeline. To do so, we first construct and add a `CHANGEPOINT` classification task to both the STL and MTL models on top of the loss, noise, and congestion tasks described previously and examine the same performance metrics after training the models. Then, we explore the performance benefits of composing existing sub-tasks in the MTL model to conduct combined network tasks as described in Chapter 2.

[4] The F1 score is defined as the harmonic mean of precision and recall, or $\frac{\boldsymbol{tp}}{\boldsymbol{tp}+\frac{\mathbf{1}}{\mathbf{2}}\cdot(\boldsymbol{fp}+\boldsymbol{fn})}$, where $\boldsymbol{tp}$ is the number of true positives and $\boldsymbol{fp}$, $\boldsymbol{fn}$ are the number of false positive and negative predictions the model makes

C. RESULTS

1) TRAINING TIMES AND MODEL ACCURACY:

When training in series on the CAIDA Ark datasets, the STL models we developed took an average of $135.4 + 96.15 + 143.4 = 374.95$ seconds to train and evaluate all three sub-tasks as shown in Table 1:. In comparison, our MTL approach took an average of 15.01 seconds (nearly 96% faster) to train and evaluate the same sub-tasks. Even if the STL models were trained in parallel, the total duration would still be equal to the greatest training time across all three tasks—in this case, 143.4 seconds, which our MTL approach outperforms by nearly 90%. Furthermore, each of the tasks in the MTL model demonstrated equal or better accuracy when compared to their STL equivalents, with an average improvement of 2.23 percentage points per task as depicted in Figure 5. The naïve methods also shown in Figure 5 depict the accuracy of the statistical classifiers described in Chapter 4, Section B. These methods were somewhat able to identify noise in the measurement data, but overall were largely unable to classify any sub-tasks accurately. Thus, due to their modest-at-best F1 scores and the fact that this behavior has been documented and reflects the findings shown in [4] and [2], we do not consider these naïve methods throughout the remainder of this paper and move to focus only on STL and MTL models for the remainder of our experiments.

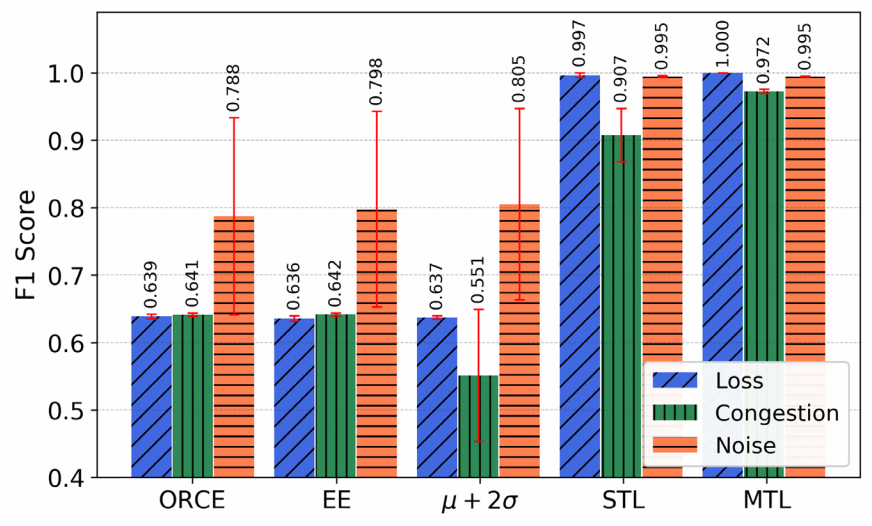


Figure 5: Average F1 Score by Subtask, comparing naïve methods, STL, and MTL approaches. Higher F1 scores are better.

Method	Task	Average Training Time	Standard Deviation
Single-Task	LOSS	135.4s	8.22s
	NOISE	143.4s	8.80s
	CONGESTION	96.15s	11.02s
Multi-Task	LOSS NOISE CONGESTION	15.01s	3.54s

Table 1: Model training times and standard deviations in seconds for all subtasks on the CAIDA Ark datasets. Lower training times are better.

2) Model Scalability:

Next, to demonstrate how ARISE facilitates improvements in model scalability, we implement an additional sub-task (`CHANGEPOINT` detection) in each model pipeline with the same hyperparameters (described in Appendix) to observe the resulting changes in performance. For `CHANGEPOINT` classification, our labeling function is as follows: we label a given measurement as a change point if the distance of the observation to the average of its neighbors exceeds a certain threshold. As the CAIDA datasets exhibit a fairly similar distribution of latency times across measurement subsets, we select a

threshold of 30ms as our changepoint indicator as it appears suitable for the distributions on which we are testing.

Given the modular nature of ARISE, the incorporation of this sub-task into the framework is straightforward. Its implementation requires only the definition of the additional labeling function, which is then passed to ARISE alongside the other labeling functions to first label the data, then subsequently create, train, and evaluate the MTL model. In contrast, the process for adding `CHANGEPOINT` classification to the STL models is much more involved. To add this sub-task, we repeat the data pre-processing and label generation steps manually and re-generate the probabilistic heuristics that the STL models employ during the training process. Even if these steps were automated to streamline the process for creating new STL classification tasks, it would still result in an STL model that suffers from the same limitations regarding the extensive amounts of time necessary to pre-process the data and train the model to a sufficient level of accuracy.

After incorporating `CHANGEPOINT` classification and retraining both STL and MTL models on the CAIDA datasets, we find that ARISE-based MTLs continue to outperform the STL approach to an even greater extent than in our initial experiments, as shown in Figure 6. More concretely, in adding this sub-task to ARISE, we find that the classification accuracy improved by an average of 12.7% when compared to that of the STL models. We also find that the F1 scores of the other tasks within the model improved slightly (by a mean of 0.7%), while the average training times increased by only 4.07 seconds (see Table 2). In comparison, the STL equivalent resulted in no improvement in the performances of other classification tasks and increased the average training time for a collection of models trained in series by 52.30 seconds with less than desirable

classification accuracy. We also note that this task employed the early stopping capabilities of the STL models during training (described in Appendix) much earlier and more frequently than other tasks due to a lack of notable performance improvements with subsequent training iterations. Thus, the resulting training times were shorter, but the classification accuracy also significantly worse. If the STL models were trained concurrently, the training duration would be equal to the length of the greatest individual training period across all tasks—in this case, 143.4 seconds, which is still significantly larger than the average training time of our MTL approach.

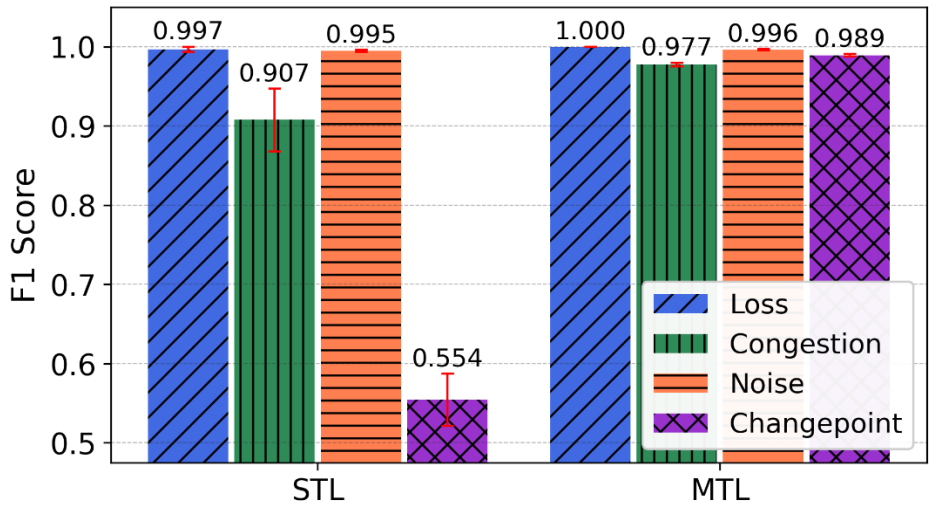


Figure 6: Average F1 score by task with the addition of `CHANGEPOINT` classification. Higher F1 scores are better.

Method	Task	Average Training Time	Standard Deviation
Single-Task	LOSS	135.4s	8.22s
	NOISE	143.4s	8.80s
	CONGESTION	96.15s	11.02s
	CHANGEPOINT	52.30s	5.84s
Multi-Task	LOSS NOISE CONGESTION CHANGEPOINT	19.08s	3.32s

Table 2: Model training times and standard deviations in seconds for all subtasks with the addition of CHANGEPOINT classification. Lower training times are better.

3) TASKS WITH MULTIPLE SUB-TASKS:

Beyond isolated feature classification, the modular nature of ARISE is also effective in enabling the construction of complex tasks involving multiple sub-tasks with composite characteristics. To demonstrate this, we combine several task-specific labeling functions to create new, specific tasks capable of identifying characteristics in conjunction with one another. These tasks use the output of relevant task heads and the insights from the shared layer to conduct their analyses. By leveraging the insights already discovered in training on relevant sub-tasks, these tasks can be added to a multi-task model with less performance overhead than an STL would incur. Consequently, the same information sharing capabilities that can enhance the accuracy of individual sub-tasks can also be used to train new classification tasks in a fraction of the time.

In contrast, single-task learning requires that we train and chain N independent STLs together (one per relevant sub-task) to fulfill a combined classification task. In this approach, the accuracy of the final classification is also effectively limited by the performance of the least accurate model in the chain (the "weakest link"). If one sub-task

fails to perform sufficiently, it is likely that the combination of tasks relying on it will be insufficient as well. Thus, we do not train an additional STL on these combined tasks, and instead consider the comparable STL performance to be the worst-case accuracy of the related tasks (i.e., the STL accuracy of **TASK 1** would be equal to the minimum across the `NOISE`, `CHANGEPOINT`, and `CONGESTION` sub-tasks, while **TASK 2** would be equal to the minimum score across the `NOISE`, `CHANGEPOINT`, and `LOSS` sub-tasks.

We model the two tasks described in Chapter 1 using ARISE to demonstrate how a network operator might interact with our framework. To reiterate, **Task 1** seeks to isolate and remove noisy data points from a series of measurements, then classify all changepoints leading to a congestion event. A single-task implementation of this would employ each of the `NOISE`, `CHANGEPOINT`, and `CONGESTION` sub-tasks separately in isolated STLs, whereas in the context of ARISE, we are able to perform this task using a total of four task heads (one for the composite task, and one for each sub-task) and the shared layers that have already been trained. To implement this, we define a labeling function l_1 that returns a positive prediction if any of the labeling functions for the related sub-tasks (l_2, l_3, and l_4, respectively) also return a positive prediction. In terms of code, we use the logical `OR` (`V`) operation between labeling functions, resulting in functionality akin to $l_1 = l_2 \vee l_3 \vee l_4$. Similarly, **Task 2** uses the same method but with different sub-tasks. That is, Task 2 employs each of the `NOISE`, `CHANGEPOINT`, and `LOSS` sub-tasks to remove noisy measurements, identify change points, and then classify instances of packet loss that follow.

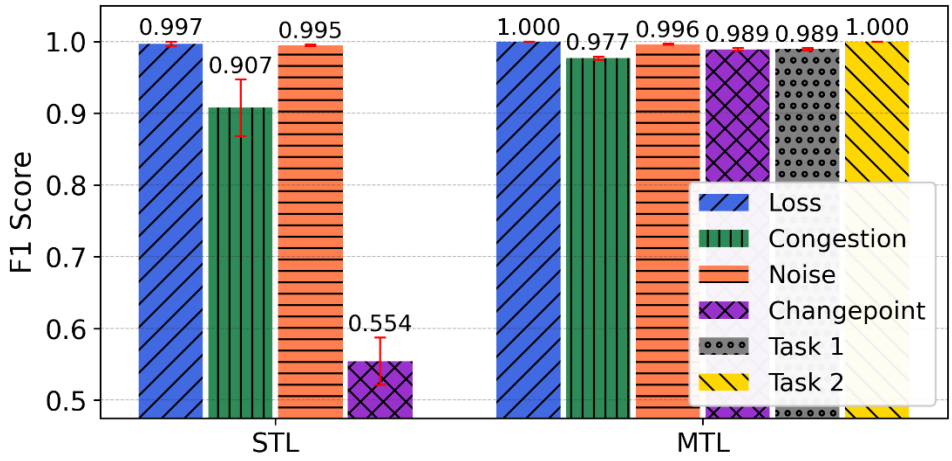


Figure 7: Average F1 scores for all Tasks and sub-tasks. The F1 scores for Tasks 1 and 2 in the STL models are equal to the worst-case score among the related sub-tasks.

Figure 7 depicts the results of composing sub-tasks across the CAIDA datasets. We see that the average F1 scores for Tasks 1 and 2 in the MTL model were 0.989 and 1.0, in contrast to the comparable STL scores, which would be $\min\{0.996, 0.554, 0.907\} = 0.554$ for Task 1 and $\min\{0.996, 0.554, 0.997\} = 0.554$ for Task 2 as well, indicating that our methods significantly outperform the STL equivalent.

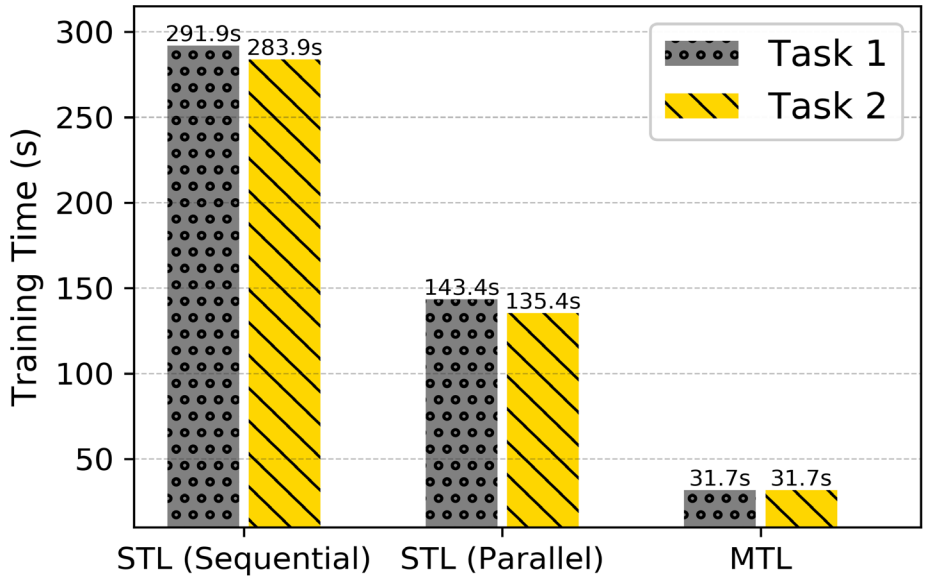


Figure 8: Average training times for composite Tasks. Lower training times are better.

To complement Figure 7, we show the resulting average training times in Figure 8 for both the STL and MTL approaches. Note that while the average training time did increase from the previous trial without composite tasks by 12.6 seconds across our experiments, the construction of similar composite tasks in STL would require the model to re-learn all of the previous characteristics exhibited by loss, noise, congestion, etc., to classify such a task. In contrast, our approach requires only that the model train one new task head to leverage the shared layer's insights, and trains much faster as a result. If trained sequentially, the average STL training time to classify these tasks would the sum of average times for each related sub-task. More concretely, the STL average training time on the CAIDA datasets would be $\sum \{\texttt{NOISE}, \texttt{CHANGEPOINT}, \texttt{CONGESTION}\} = 96.15 + 52.3 + 143.4 = 291.85$ seconds for Task 1 and $\sum \{\texttt{NOISE}, \texttt{CHANGEPOINT}, \texttt{LOSS}\} = 96.15 + 52.3 + 135.4 = 283.85$ seconds for Task 2. If the STL training were parallelized (meaning each task trained concurrently), the resulting training times for the composite tasks would be equal to the worst-case time of each of the related tasks (i.e., the maximum time from the `NOISE`, `CHANGEPOINT`, and `CONGESTION` sub-tasks for Task 1 and the `NOISE`, `CHANGEPOINT`, and `LOSS` sub-tasks for Task 2).

We show the theoretical results of this parallelization in Figure 8 alongside the MTL and sequential STL models. Though this parallelization may result in faster training times for STL models, these models do not possess the information sharing capabilities that MTL classifiers do and thus fail to attain the same level of classification accuracy. In both cases, our MTL approach trains significantly faster on a larger number of tasks as it effectively leverages the shared layer and composite characteristics to train new tasks with reduced performance overhead.

CHAPTER 5: DISCUSSION AND FUTURE WORK

A. HOW MIGHT ONE USE ARISE FOR DIFFERENT NETWORKING TASKS?

We prescribe the following steps for researchers and/or network operators who may wish to use ARISE to generate a multi-task model for different networking tasks (e.g., other than the ones presented in this work). We illustrate the steps with an example of an amplification-style DDoS attack detection task.

- Identify a data source (e.g., NetFlow) and generate a time series for the network features of interest such as *heavy hitters* (e.g., number of flows per second, unique number of source IP addresses, etc.).
- Translate the known domain heuristics into concrete labeling functions that can be applied as code. For example, if the number of flows per second $> A$ and number of sources $> B$, then we are seeing an onset of a DDoS attack.
- Construct the task heads and shared layers, employing either complete information sharing (as discussed in Chapter 3, Section C) or some alternative information sharing paradigm. For example, all attacks share heavy hitters-based features (which can be used to create shared layers), but their protocol-specific signatures are different and can be used to train attack-specific head layers (e.g., ICMP vs. NTP vs. DNS).

B. FUTURE WORK.

We plan to expand the capabilities in ARISE and investigate the following problems:

1) MODERNIZING NETWORK MANAGEMENT WITH ARISE:

We believe ARISE offers a new way to manage networks in the age of AI/ML: through a *conversational intelligence approach to network management*. In this

approach, an operator will be able "chat" directly with a network management agent that can then translate the operator's tasks with natural language processing (NLP) techniques (e.g., [40], [41]) and train/launch appropriate ARISE-based MTL models for diverse performance or security-related tasks. While a narrow version of this approach is already supported in ARISE, we intend to investigate the end-to-end design of such as approach by leveraging existing NLP techniques [40], [41] and designing a spectrum of network task examples as part of future work.

2) ENHANCING THE ROBUSTNESS OF ARISE WITH ADVERSARIAL EXAMPLES:

Gaining an operator's trust also hinges on the extent to which a model's predictions are right (or wrong) when compared with the ground truth. To understand this issue, we plan to investigate a training method that can consume a small, labeled dataset from ARISE and a large unlabeled dataset to produce "safe" models (i.e., learning models that can account for low-probability/high-impact input examples). The method concerns the generation of adversarial examples—intentional feature perturbations that can cause a model to make false predictions—using e.g., [42] to create such examples. The goal is to produce one set of examples that are based on predicted labels and another set of examples that are based on predicted labels with feature perturbations and use them to quantify the "extent" to which the predictions are wrong compared to ground truth.

CHAPTER 6: SUMMARY

The use of ML for networking faces several obstacles including lack of labeled datasets, scarcity of operators' domain expertise during the labeling process, multiplicative model training times with increasing number of network tasks, and inability to reason about decisions of the models. In this paper we propose ARISE, a novel framework to overcome those obstacles. ARISE is capable of leveraging innovations in weak supervision and multi-task learning and operator expertise in the form of labeling functions to address prior limitations of ML efforts in the field by enhancing model accuracy, scalability, and dramatically reducing training times.

We demonstrate the efficacy of ARISE by considering different tasks such as inferring loss, congestion, change points, and noise, and comparing the performance of ARISE-based MTL vs. STL models in the context of CAIDA's Ark datasets. Our results show that, in addition to outperforming naïve statistical methods, MTL models can be trained up to 8x faster with over 40% accuracy improvements as measured by F1 score when compared to STL models.

APPENDIX A: MODEL TRAINING AND TUNING

To train our classification models, we first establish a set of *functional hyperparameters* for STL and MTL. These parameters influence the underlying behavior of a model, and include components such as the learning rate, number of training iterations to perform, and any other technical optimizations one may wish to implement. In each of our experiments, we train both the STL and MTL models repeatedly with different sets of hyperparameters (i.e., varied learning rates) before the selecting the final parameters and evaluating the model. After experimenting with a number of different learning rates for each approach, we select a static rate of 0.005 for each of the STL models and an initial learning rate of 0.01 in the MTL approach. In training the MTL models, we found that the use of a constant learning rate occasionally resulted in increasingly volatile behavior in the task heads. To address this, we implement a simple learning rate scheduler that linearly decreases the current rate with each subsequent training iteration. This results in the MTL model's predictions being much more accurate and consistent overall. Similarly, in deciding the number of training iterations to conduct, we evaluate the resulting model performances for a number of epochs before ultimately choosing to perform 20 training iterations for the STL models and 10 for the MTL approach.

Each model also requires a scoring metric to evaluate its performance after the conclusion of the training process. We use the F1 score of each model as this numerical measure, providing equal weight to the importance of both precision and recall in evaluating our models. We calculate and report the average of F1 scores across the subsets of data. To train effectively, each model also requires a *loss function* that will be used to

evaluate the predictions of the current training iteration with respect to the intended or "ground truth" output. For STLs, we use *binary cross entropy* as our standard loss function to reflect the STLs' ability to produce only True/False predictions. For MTLs, we use a *generic cross entropy* loss function to better support future ARISE-based efforts that may wish to employ more than simple binary classification on a single task (i.e., enabling models to vote `ABSTAIN` for a certain input if insufficient evidence is presented in training, or combining predicted labels for more complex feature classification).

We also employ various optimization strategies to further enhance the performance of the STL models such as L2 regularization, dropout, and early stopping to enable the early termination of the training process when a lack of performance improvements from subsequent iterations is detected [2]. In these models, early stopping is triggered when the model's loss function fails to decrease by at least 0.01 over the course of five subsequent training iterations. This optimization also means that the number of STL training iterations actually performed is often far fewer than 20 epochs (as indicated above), with the number of observed iterations frequently ranging approximately between 9 and 13 (compared to 10 epochs in our MTL approach). We intentionally chose not to implement similar optimizations in our MTL approach both for the sake of simplicity and to show how MTL compares against a more complex and better optimized combination of STL models.

BIBLIOGRAPHY

[1] J. Zerwas *et al.*, "NetBOA: Self-Driving Network Benchmarking," in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, 2019, pp. 8–14.

[2] Y. Lavinia, R. Durairajan, R. Rejaie, and W. Willinger, "Challenges in Using ML for Networking Research: How to Label If You Must," in *Proceedings of the Workshop on Network Meets AI & ML*, New York, NY, USA, 2020, pp. 21–27. doi: 10.1145/3405671.3405812.

[3] "K. Claffy, D. Clark, F. Bustamante, J. Heidamann, M. Jonker, A. Schulman, and E. Zegura. Workshop on Overcoming Measurement Barriers to Internet Research (WOMBIR 2021) Final Report." [Online]. Available: https://www.caida.org/catalog/papers/2021_wombir2021_report/wombir 2021_report.pdf

[4] A. Muthukumar and R. Durairajan, "Denoising Internet Delay Measurements using Weak Supervision," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, 2019, pp. 479–484.

[5] A. Gupta, C. Mac-Stoker, and W. Willinger, "An Effort to Democratize Networking Research in the Era of AI/ML," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, 2019, pp. 93–100.

[6] J. Baxter, "A model of inductive bias learning," *Journal of artificial intelligence research*, vol. 12, pp. 149–198, 2000.

[7] "CAIDA Ark Datasets." [Online]. Available: http://www.caida.org/projects/ark/topo_datasets.xml

[8] K. Xu and J. Chandrashekar and Z.L. Zhang, "A First Step toward Understanding Inter-domain Routing Dynamics," 2005.

[9] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76.

[10] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification," *ACM SIGCOMM CCR*, 2006.

[11] A. McGregor, M. Hall, P. Lorier, and J. Brunskill, "Flow clustering using machine learning techniques," in *PAM*, 2004, pp. 205–214.

[12] G. Comarela, R. Durairajan, P. Barford, D. Christenson, and M. Crovella, "Assessing Candidate Preference through Web Browsing History," 2018.

[13] D. Wang, Y. Yu, and A. Rinaldo, “Optimal change point detection and localization in sparse dynamic networks,” *The Annals of Statistics*, vol. 49, no. 1, pp. 203–232, Feb. 2021, doi: 10.1214/20-AOS1953.

[14] C. Truong, L. Oudre, and N. Vayatis, “Selective review of offline change point detection methods,” *Signal Processing*, vol. 167, p. 107299, 2020, doi: https://doi.org/10.1016/j.sigpro.2019.107299.

[15] R. Killick, P. Fearnhead, and I. A. Eckley, “Optimal Detection of Changepoints With a Linear Computational Cost,” *Journal of the American Statistical Association*, vol. 107, no. 500, pp. 1590–1598, Oct. 2012, doi: 10.1080/01621459.2012.737745.

[16] A. Celisse, G. Marot, M. Pierre-Jean, and G. J. Rigaill, “New efficient algorithms for multiple change-point detection with reproducing kernels,” *Computational Statistics & Data Analysis*, vol. 128, pp. 200–220, 2018, doi: https://doi.org/10.1016/j.csda.2018.07.002.

[17] J. Bai, “Estimating Multiple Breaks One at a Time,” *Econometric Theory*, vol. 13, no. 3, pp. 315–352, 1997, doi: 10.1017/S0266466600005831.

[18] P. Fryzlewicz, “Wild binary segmentation for multiple change-point detection,” *The Annals of Statistics*, vol. 42, no. 6, Dec. 2014, doi: 10.1214/14-aos1245.

[19] P. Fryzlewicz, “Unbalanced Haar Technique for Nonparametric Function Estimation,” *Journal of the American Statistical Association*, vol. 102, no. 480, pp. 1318–1327, 2007, doi: 10.1198/016214507000000860.

[20] E. J. Keogh, S. Chu, D. M. Hart, and M. J. Pazzani, “An online algorithm for segmenting time series,” *Proceedings 2001 IEEE International Conference on Data Mining*, pp. 289–296, 2001.

[21] H. Ringberg, A. Soule, J. L. Rexford, and C. Diot, “Sensitivity of PCA for traffic anomaly detection,” in *SIGMETRICS’07 - Proceedings of the 2007 International Conference on Measurement and Modeling of Computer Systems*, 2007, 1st ed., no. 1, pp. 109–120. doi: 10.1145/1269899.1254895.

[22] A. Lakhina, M. Crovella, and C. Diot, “Diagnosing Network-Wide Traffic Anomalies,” in *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, New York, NY, USA, 2004, pp. 219–230. doi: 10.1145/1015467.1015492.

[23] A. Lakhina, M. Crovella, and C. Diot, “Mining Anomalies Using Traffic Feature Distributions,” in *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, New York, NY, USA, 2005, pp. 217–228. doi: 10.1145/1080091.1080118.

[24] J. Camacho, A. Pérez-Villegas, P. García-Teodoro, and G. Maciá-Fernández, "PCA-based multivariate statistical network monitoring for anomaly detection," *Computers & Security*, vol. 59, pp. 118–137, 2016, doi: https://doi.org/10.1016/j.cose.2016.02.008.

[25] X. Li *et al.*, "MIND: A Distributed Multi-Dimensional Indexing System for Network Diagnosis," 2006.

[26] M. Syamkumar, S. K. Mani, R. Durairajan, P. Barford, and J. Sommers, "Wrinkles in Time: Detecting Internet-wide Events via NTP," 2018.

[27] A. W. Moore and D. Zuev, "Internet traffic classification using Bayesian analysis techniques," 2005.

[28] R. Bunescu and R. Mooney, "Learning to extract relations from the web using minimal supervision," 2007.

[29] M.-C. Yuen, I. King, and K.-S. Leung, "A survey of crowdsourcing systems," 2011.

[30] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, "Holoclean: Holistic data repairs with probabilistic inference," *VLDB Endowment*, 2017.

[31] A. Ratner, S. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré, "Snorkel: Rapid Training Data Creation with Weak Supervision," *Proceedings of the VLDB Endowment*, vol. 11, no. 3, pp. 269–282, 2017.

[32] A. J. Ratner, C. M. De Sa, S. Wu, D. Selsam, and C. Ré, "Data programming: Creating large training sets, quickly," in *Advances in neural information processing systems*, 2016, pp. 3567–3575.

[33] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why Should I Trust You?': Explaining the Predictions of Any Classifier," *CoRR*, vol. abs/1602.04938, 2016.

[34] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. 2017, pp. 4765–4774.

[35] O. Bastani, C. Kim, and H. Bastani, "Interpreting Blackbox Models via Model Extraction," *CoRR*, vol. abs/1705.08504, 2017.

[36] M. Luckie, "Scamper: a scalable and extensible packet prober for active measurement of the internet," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010, pp. 239–245.

[37] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, “Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package),” *Neurocomputing*, vol. 307, pp. 72–77, 2018, doi: https://doi.org/10.1016/j.neucom.2018.03.067.

[38] A. Ratner, B. Hancock, J. Dunnmon, R. Goldam, and C. Ré, “Snorkel MeTaL: Weak Supervision for Multi-Task Learning,” *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, 2018.

[39] CloudLab.us, 2021. [Online]. Available: https://www.cloudlab.us/

[40] A. Alsudais and E. Keller, “Hey network, can you understand me?,” in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2017, pp. 193–198.

[41] R. Birkner, D. Drachsler-Cohen, L. Vanbever, and M. Vechev, “Net2Text: Query-guided summarization of network forwarding behaviors,” in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 609–623.

[42] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, and D. Song, “Generating Adversarial Examples with Adversarial Networks,” *CoRR*, vol. abs/1801.02610, 2018, [Online]. Available: http://arxiv.org/abs/1801.02610