ARISE: A Multi-Task Weak Supervision Framework for Network Measurements

Jared Knofczynski, Ramakrishnan Durairajan (University of Oregon), Walter Willinger (NIKSUN, Inc.)

***Abstract*—The application of machine learning (ML) to mitigate network-related problems poses significant challenges for researchers and operators alike. For one, there is a general lack of labeled training data in networking, and labeling techniques popular in other domains are ill-suited due to the scarcity of operators' domain expertise. Second, network problems are typically multi-tasked in nature, requiring multiple ML models (one per task) and resulting in multiplicative increases in training times as the number of tasks increases. Third, the adoption of ML by network operators hinges on the models' ability to provide basic reasoning about their decision-making procedures.**

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 learning paradigms such as multi-task learning (MTL) and meta-learning to facilitate information sharing between tasks as well as reduce overall training time. Using community datasets, we show that ARISE can generate MTL models with improved classification accuracy compared to multiple single-task learning (STL) models. We also report findings that show the promise of MTL models for providing a means for reasoning about their decision-making process, at least at the level of individual tasks.

I. Introduction

THE application of machine learning (ML) to the field of networking has received mixed reactions from the network research and operator communities. On the one hand, there continues to be great interest and enthusiasm among network researchers in developing new learning models for an increasing number of diverse network management tasks, be they performance- or security-related [1]. However, this enthusiasm has been hampered by a general lack of data (especially *labeled* training data), has generated little to no interest in demonstrating whether or not a trained ML model will generalize as expected in deployment scenarios (i.e., can it be trusted in practice), and has resulted in the development of ever more complex black-box learning models without much or any concern for attempting to explain how they "work" (i.e., make their decisions) [2]. 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 often prefer to rely on more traditional statistical techniques in conjunction with manual oversight [4], resulting in increased operator workloads.

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, (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 (iii) the continued focus on black-box learning models which are incompatible with operators' need to "trust" the inferences made by developed learning 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 simple scenario that we will use throughout the paper as an illustrative example. 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., remove noise, identify changepoints, detect congestion in the case of Task 1; and remove noise, identify changepoints, and identify 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. Last but not least, the quest for developing trust in an ML model requires being able to reason about its decisions or inferences, at least at the level of individual sub-tasks.

To address the aforementioned 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 called ARISE[1]. On the one hand, ARISE allows network operators to leverage their domain expertise in the form of labeling functions (i.e., programmatic representations of domain knowledge and heuristics related to specific networking tasks). This way, operators are able to automatically label large training data sets *without* the need for any crowdsourcing or manual labeling that may be impractical in the networking domain. At the same time, by relying on multi-task learning (MTL), ARISE enables related network classification tasks to train concurrently, thereby expediting the training process and improving the classification accuracy of related tasks through information sharing in the hidden layers. In this sense, ARISE has much in common with existing alternative learning paradigms such as meta-learning [5]. Lastly, ARISE also enables network operators to begin reasoning about the decisions made by their ML models, enabling further trust on the behalf of network operators in their models as a result of their ability to interpret the reasoning behind their models' decisions. In short, ARISE provides a practical framework for supporting the fast and efficient training of multiple ML models in ways that exploit the commonality of distinct tasks without reducing accuracy and afford 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 issues that existing STL models and statistical methods face. These studies often involve 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 [6], the application of scalable techniques to overcome the data labeling problem [2], and the development of customized models for individual network management tasks (see § II-B).

But unlike these state-of-the-art efforts, ARISE goes beyond STL models and makes the following novel contributions:

- ARISE leverages shared layers and uses individual task-specific head layers in model training. The former capture commonalities among features across different tasks in a principled manner using MTL and are trained only once. The latter provide a means for adding new tasks "on the fly" in a way that only requires training the head layers. MTL has already proven to generalize better than STL models due to *implicit data augmentation* [7]; i.e., the sharing of information across tasks in the model resulting in an enhanced availability of data on which to train.
- As a framework at the intersection of multi-task and meta-learning, ARISE combines the best of both worlds; i.e., fast adaptation to new unseen tasks with efficient training of new models. As such, it addresses limitations that restrict the use of many ML methods in practice (especially for networking tasks), and we provide concrete examples on how ARISE overcomes these limitations.
- ARISE can succinctly capture domain knowledge in three 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 novel 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.
- More broadly, while data programming and MTL are well-studied and the availability of their implementations notwithstanding, our key contribution is in the adaption of these ideas to the application domain of networking. In doing so, we solve several domain-specific adaptation challenges including data imbalance issues, support for time series datasets, and network task-specific feature extraction from those datasets. None of these aspects are available in existing implementations of data programming and MTL.

To evaluate ARISE, we compare the F1 scores and training times of MTL and STL models across different sub-tasks such as loss, noise, congestion, and changepoint classification using CAIDA's Ark and RIPE's Atlas datasets [8], [9]. We also compare the resulting F1 scores of both STL and MTL models to similar classification tasks conducted via traditional statistical techniques and demonstrate that these methods are often 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 than STLs and exhibit reduced training overheads when implementing new sub-tasks, facilitating model scaling. Moreover, MTLs 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.

II. Motivation and Background

With growing traffic volumes and continued innovations in today's networks, 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 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 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 above. Both tasks are of potential interest to network operators and researchers alike, and both tasks are inherently dependent on multiple sub-tasks (e.g., **Task 1** involves identifying and

[1]The source code for ARISE can be found at https://gitlab.com/onrg/arise.

removing noise present in the input data, identifying network congestion events, and detecting subsequent changepoints in the latency measurements).

The aforementioned characteristics can appear in many different forms, and there is often a significant degree of overlap between these characteristics across sub-tasks. We refer to these overlapping features as *composite characteristics*. 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 [8]; more details in § IV-A) similar to that which a network operator or researcher might observe. Within the depicted period, we hypothesize two distinct network events (noise and congestion) that an operator or researcher may wish to detect.

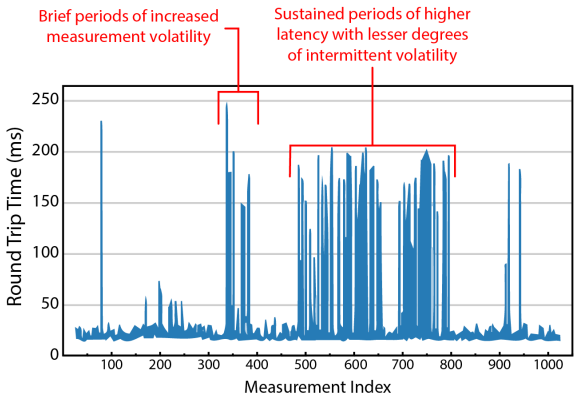


Figure 1: Visualization of composite characteristics in time series network data.

On the left of Figure 1 (between indices 300 and 400), we observe 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 momentary 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 brief period of volatility, a similar pattern occurs 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 might result in additional latency in communications between network hosts.

While there are no commonly agreed upon specific criteria for classifying network features such as measurement noise [4], underlying attributes such as network volatility are commonly shared between distinct network features that an operator may wish to classify. And as the classification of these features relies so heavily on the identification and classification of such shared components, the information sharing capabilities of MTL make ARISE a promising tool to begin classifying (and distinguishing between) network features such as loss, noise, and congestion.

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 problems in a network. Each of these classification tasks relies to some extent on the interpretation of variation between 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.[2] Thus, we posit that the information sharing capabilities of MTL will allow for ARISE to more effectively classify network features such as loss, noise, changepoints, and congestion, in comparison to prior efforts.

2) Model scaling is impossible with single-task learning (STL): 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 for all STLs in a given task is equal to the longest training time among the individual STL models.

Beyond simply increasing the amount of training time required for holistic model development, the lack of information sharing in STLs—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 model effectively discards any information it gleans during its training process that it deems not relevant to 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., [10]). In other words, when a given model returns a specific decision, it is generally impossible for the network

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

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. Prior Efforts and their Limitations

Applying ML to solve networking problems is of great importance to the community, 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) [11], [12]. Complementary to these techniques are statistical methods such as expectation maximization [13], [14], with one such naïve method being the use of simple filters such as $\mu+3\sigma$ to classify latency measurements as noise [4]. Other examples include changepoint classification (e.g. [15]), which allow operators to identify changepoints in recorded data. Typical shortcomings of these techniques are (i) their inability to capture operators' domain knowledge, (ii) their inherent need for labeled datasets that are hard to come by in the networking domain, and (iii) their difficulties in accurately quantifying changepoints and other network features without significant operator input. Existing methods for changepoint classification also require that the entire collection of time-series data being analyzed be gathered and known beforehand. As a result, they must operate offline and could therefore not be used effectively in online, real-time network management scenarios. See Appendix B for details on the shortcomings of naïve methods for changepoint classification in time series measurement data.

The second category of efforts are unsupervised learning techniques such as clustering and Principal Component Analysis (PCA) to detect anomalies in network data (e.g., BGP [10] and traffic measurements [16], [17], [18], [19]), help with network diagnosis [20], and perform event detection [21]. These efforts suffer from a general inability to interpret their decisions or reason about their underlying decision-making processes, and as such, pose a significant obstacle with respect to the adoption of ML-based methods to address networking problems. For example, Ringberg et al. [16] 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 efforts 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 [22], [23] and crowdsourcing [24], [25], both of which suffer from issues such as inaccuracy and inadequate coverage. Addressing these issues is the dual objective of Snorkel [26], 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 [27]. Inspired by this data programming paradigm, two recent efforts (EMERGE [2] and NoMoNoise [4]) seek to classify network features using weak supervised learning. However, these efforts in their current form have limitations that restrict their use in practice. For one, while effective in classifying measurement noise, neither effort is 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 the local (e.g., LIME [28], SHAP [29]) or global (e.g., Bastani et al. [30]) levels. ARISE builds on these prior works and leverages classification methods from both EMERGE and NoMoNoise in both the STL and statistical models (respectively) that we employ to evaluate ARISE.

In addition to efforts in networking, recent endeavors in the field of ML have attempted to leverage MTL for several widely varied application domains such as intrusion detection [31], data augmentation for convolutional neural networks [32], HIV therapy screening [33], etc. While such efforts are promising in their respective fields, to the best of our knowledge, ours is the first effort to apply MTL to networking tasks.

Also relevant in this context are prior efforts regarding other related learning paradigms such as *meta-learning*, or *learning to learn*, which attempts to leverage common representations of underlying information to quickly adapt existing models to novel tasks [34]. While the meta-learning paradigm appears promising in theory, there are several aspects that make it impractical for applied use in the context of networking. For one, previous efforts involving meta-learning (such as [35], [36], [37]) have typically relied on large, labeled datasets (e.g., Omniglot [35], ImageNet [38], and MNIST [39]), but such types of datasets are generally unavailable in the networking domain. Furthermore, prior efforts exploring the intersection of meta-learning and MTL have shown that gradient-based meta-learning algorithms can be much more costly and time-consuming to train when compared to MTL classifiers [5]. Thus, while learning paradigms such as meta-learning are promising in their ability to generalize to novel tasks with minimal examples, ours is the first effort to practically leverage meta-learning to further enhance the application of MTL for networking tasks.

Note that meta-learning also refers to any underlying algorithms or model structures that assist an ML model in learning more effectively, such as by leveraging AutoML [40] techniques to select optimal model hyperaparameters with minimal operator effort. Hyperparameter optimization (HPO) strategies can be an effective tool in the construction of ML models that are both fair (with respect to the range of tasks being trained in a given model) and resilient. Here, an ML

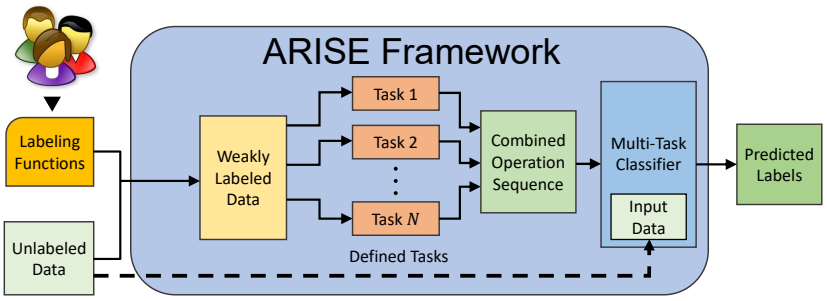


Figure 2: Design of the ARISE framework.

model is called resilient if it is largely immune to variations in task specification and the underlying data distributions associated with specific features in distinct datasets such as those found across applications of ML models in network measurements. However, HPO alone cannot effectively leverage information sharing or composite characteristics to improve model performance beyond what would be possible for a conventional STL model. Thus while HPO and other meta-learning strategies allow for significant potential improvements of model performance compared to more traditional efforts, both require additional insight before they can be leveraged to their fullest extents for networking tasks. (See § IV-D for an empirical investigation into the efficacy of prior meta-learning efforts vs. our MTL approach.)

III. Design and Implementation of ARISE

In this section we provide an overview of ARISE, describe the design details of its individual components, and present the technical specifications of its 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 overcomes some of the aforementioned limitations. ARISE is novel in that it 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. ARISE also leverages ideas from meta-learning to efficiently train models while allowing models to scale to new unseen tasks, providing a practical demonstration of theoretical guarantees shown in [5]. While each of these ideas have been utilized 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 three key benefits, each of which will be described in detail in § III-B below:

- The resulting MTL models 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 meta-learning capability facilitates model scaling, drastically reducing model training times.
- Using 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 previously). ARISE relies on a multi-task classification technique that allows the model to generalize better due to *implicit data augmentation*. This technique also ensures the accuracy of the model.
- ARISE is capable of leveraging available domain knowledge at several stages in its design. These stages refer to the use of labeling functions (i.e., simple programmatic representations of operator know-how), the creation of the shared layers, and the exploitation of relationships between tasks to improve the performance of prior stages. Including such domain knowledge in the design of ARISE enables operators to qualitatively reason about the decisions taken by MTL models at the task level, thus lowering the adoption barrier for network operators.

By incorporating these insights, ARISE thus allows operators to begin leveraging their domain expertise to label data at scale, reasoning about the use of composite characteristics to improve classification accuracy, and examining the reasoning behind model predictions. We are not aware of other efforts in the context of networking that give operators the capabilities provided by and implemented in ARISE.

B. Design Details

Figure 2 depicts the overall architecture of ARISE framework. We describe each of the components below.

User interface. At its core, the success of ARISE is dependent on its ability to capture and translate the domain expertise of network operators into concrete, tangible heuristics expressed as literal lines of code (e.g., if a latency measurement x is greater than the average $\mu + 3\sigma$ for a given time series, then x is a noisy measurement). To this end, we have designed a simple interface that operators and researchers can use to convert their domain knowledge regarding network events into programmatic representations (e.g., using Python) referred to as labeling functions.

Unlabeled data. Input data for ARISE can be collected using traditional measurement techniques (e.g., `scamper` [41]) or based on other vendor-specific representations (e.g., NetFlow). 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 commonly used to provide operators with a temporal view of their data so they can assess networking problems at scale (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. Borrowing the idea of data programming (e.g., see Ratner et al. [27]), operators can construct simple labeling functions such as those shown above to generate weak labels for large, unlabeled networking datasets that can be used in ML model training.

In this paper, we use *tsfresh* [42] to extract feature-specific threshold values for each dataset such as those used in [2], [4]. We demonstrate how these thresholds can be used to label data points with the help of the noise labeling function `label_noise` shown above (where `NORMAL` and `VOTE` correspond to the integer values 0 and 1). This function labels each point as noise or normal while still accounting for the baseline differences in each dataset's individual behavior. More concretely, this labeling function labels a measurement as noise if its latency falls in the range $[1.5 \cdot \alpha, \infty)$, where α is the value of the RTT at the 75$^{\text{th}}$ percentile for the current dataset. A similar function can be used to label a point as `CONGESTION` if the round-trip time of the input measurements falls in the interval $[1.2 \cdot \beta, 1.5 \cdot \alpha)$, where β is the RTT value at the 30$^{\text{th}}$ 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 0 indicates that a packet was lost.

We select the thresholds for these labeling functions by manually examining several of the input datasets to determine an appropriate latency range for each feature being classified. We note that while these thresholds may change depending on the specific application contexts and datasets on which ARISE is implemented, other users of ARISE may easily develop their own labeling functions by leveraging their knowledge of the data and translating their domain heuristics into concrete labeling functions such as the label_noise and label_congestion functions shown above. As labeling functions are intended to distinguish between characteristics based on an operator's understanding of the underlying data, they can (and should) be adapted for specific contexts based on an operator's familiarity with a 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, similar to what they would do for a traditional ML model. In the case of ARISE, as there are no definitive "ground truth" labels with which to compare, the weak labels generated by the labeling functions also serve as the ground truth with which ARISE 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, tasks such as **Task 1** or **Task 2** (in § I) entail multiple sub-tasks, and each sub-task corresponds to a specific task head in the underlying model which is responsible for leveraging the insights of the shared layer to perform feature classification on the input data. For example, a model seeking to perform **Task 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 head layers are defined, we merge them into a combined operation sequence that is specific to the Snorkel library [43] and acts as a logical container for the individual classification elements in the MTL model.

Combined operation sequence. The task heads in ARISE each 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 § II-A1). 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. As a result, the shared layer is able to learn both the individual and composite characteristics that comprise the features being classified. It 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 design 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 other tasks by learning how other tasks' predictions correlate with their own.

This implicit data augmentation reduces the risk of underfitting a model to the tasks at hand. Indeed, 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 any restrictions posed by other task heads. By the same logic, since the MTL classifier must optimize its performance with respect to each of its classification tasks during training, implicit data augmentation also reduces the risk of overfitting the model to any one specific task.

Alternate implementations of information sharing, other than complete sharing, is also possible in ARISE. For example, by specifying exactly which layers of the MTL model share information with one another, one may choose to isolate specific relationships between network characteristics or remove the influence of two unrelated, dissimilar tasks from one another in the training process. After the model structure has been established, the operation sequence and list of tasks can be passed to the multi-task classifier to train the final model.

Multi-task classifier. After applying weak labels to the data, constructing individual task heads, and architecting the information sharing capabilities of the shared layer, all that remains is to train the final model. To select hyperparameters such as the learning rate for a given dataset and training iteration, we leverage insight from meta-learning techniques such as AutoML [40] to select the optimal hyperparameters for a given dataset on the tasks at hand. See Appendix C for selection and tuning of model hyperparameters.

After establishing the model's hyperparameters 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 provided labeling functions. At this stage, users may also define new tasks and insights by providing additional labeling functions or modifying the underlying structure 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 model architecture in a principled and efficient manner.

We note that unlike prior multiclass classification efforts, the use of a multi-task classifier has clear advantages in that it allows us to train multiple related yet distinct sub-tasks concurrently, and thus classify measurements exhibiting multiple features at the same time (e.g., noise and congestion). This is essential when working in the networking domain, as features such as the ones we are classifying often overlap in terms of the given measurements they affect. In contrast, a multiclass classification system would allow at most one feature to be identified for each given measurement, thereby nullifying any of the benefits that information sharing provides.

C. Implementation

To implement ARISE, we build upon the key elements and weak supervision components employed in NoMoNoise [4] and EMERGE [2], and MTL components from Ratner et al. [44].[3] From each dataset, we extract a number of features and time series characteristics using *tsfresh* [42] that we then employ as thresholds in the development of our labeling functions (in § III-B). 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 final classification. This MLP module allows each task to 'eavesdrop' on the others, enabling them to learn from other 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, when the `CONGESTION` classifier attempts to identify similar characteristics, it is able to leverage the same insights from the shared 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 [43] 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 which we evaluate ARISE against employ two hidden layers *per task*; we find that incrementing the number of hidden layers in the STL models further offers little to no benefit in the resulting accuracy, and the additional layers tend to increase training times significantly. We also evaluate several optimization and pre-processing strategies, which we discuss in Appendix C. Overall, 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 § IV-C.

We base the single-task component of our analysis on the EMERGE framework described in [2], which is intended to perform weak supervision in the context of a single-task noise classification. We modify this framework to include additional capabilities such as congestion, outages, and changepoint detection by writing new labeling functions. We also evaluate the efficacy of statistical classifiers such as those used in EMERGE [2] and NoMoNoise [4] in comparison to ARISE. All models were trained and evaluated on a CloudLab.US [45] server running Ubuntu 16.04 LTS with an Intel Xeon D-1548 64-bit processor and 8GB of memory.

IV. EVALUATION OF ARISE

In this section, we describe our datasets and the experiments we conduct to evaluate the ARISE framework and compare it against alternative learning methods.

A. Datasets Used

In our experiments, we use `traceroute` and `ping` datasets from the CAIDA Ark [8] and RIPE Atlas [9] projects, respectively. From the CAIDA dataset, we select 28 distinct network source/destination pairs from the `atl2-us` vantage point located in Tucker, Georgia. These measurements span 24 hours on January 1, 2019, and contain a total of 75,359 latency measurements which serve as training sets in our

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

evaluation.[4] Individually, these contain between 2,000 to 4,000 measurements, with an average of approximately 2,700 round-trip time measurements per source/destination pair.

For the RIPE Atlas data, we select a series of 2.5 million latency measurements from the 24.5 million entries in the original dataset.[5] These measurements contain source and destination IP addresses, average round-trip times, and measurement timestamps. We split them into 100 time-series training subsets containing 25,000 latency measurements each. We use the RIPE dataset as an example of a similar, yet significantly more complex alternative to the CAIDA traceroute data to examine the efficacy of ARISE in the context of larger and noisier datasets. Note that the RIPE Atlas datasets contain measurements from multiple hosts to multiple destinations, resulting in much noisier traffic from a large number of different flows rather than a single flow. We average the training times and F1 scores of each model on both datasets in the evaluations below, and compare the results of increasing both the quantity of data and noise contained within it for the STL and MTL approaches.

B. Experiments

To evaluate ARISE, we conduct several experiments and report on (i) the efficacy of our MTL models against prior STL efforts (in terms of time, accuracy, and scalability to novel sub-tasks), (ii) the effectiveness of our MTL approach in comparison to existing meta-learning methods, and (iii) the ease-of-use of ARISE when applied in practice.

Initially, we train three STL models based on the EMERGE framework [2] to individually classify latency measurements in a network time series as either `LOSS`, `NOISE`, or `CONGESTION`. We use the output of our original labeling functions as our models' ground truth under the assumption that operators can define their own labeling functions using their domain expertise to accurately represent the characteristics of the features they wish to classify. We also train MTL models and compare their efficacy with three naïve statistical classifiers employed by prior efforts such as EMERGE [2] and NoMoNoise [4] (i.e., Overly Robust Covariance Estimation or ORCE, Elastic Ellipse or EE, and $\mu+2\sigma$). As tools often used by network operators [2], these naïve classifiers also serve as a baseline in the absence of ground truth data.

We record the times taken for each model to train, as well as the resulting F1 scores (defined as the harmonic mean of precision and recall, or

$$F_1 = \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})}$$

where **tp** is the number of true positives and **fp**, **fn** are the number of false positive and negative predictions the model makes). 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 the time-series in which we seek to classify them. We then conduct the same analyses via the multi-task approach, using the same labeling functions and evaluation metrics within ARISE, recording the model training times and F1 scores upon completion. In both cases, our selection of model hyperparameters (e.g., learning rate) is informed by meta-learning techniques such as AutoML [40], discussed in Appendix C.

To demonstrate the scalability of ARISE, we examine the effects of adding new classification tasks into the pipeline. In particular, we 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. We then compare the efficacy of meta-learning methods such as ensemble learning to that of our MTL models and report on the performance of ARISE as informed by such meta-learning techniques. Lastly, we explore the performance benefits of composing existing sub-tasks in the MTL model to conduct combined network tasks as described in § II and demonstrate the interpretative capabilities of ARISE as they could be leveraged in an applied context. In each case, we conduct the same experiments using both the CAIDA and RIPE datasets.

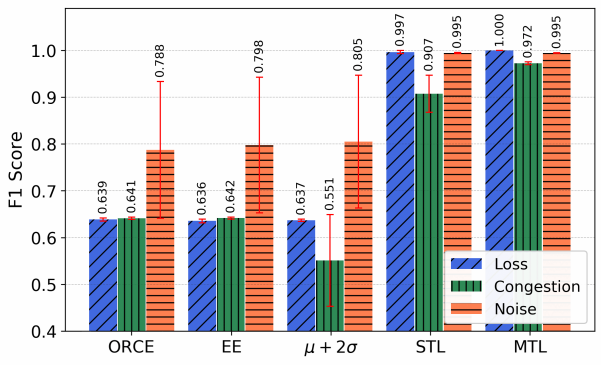
C. STL vs. MTL

1) Training Times and Model Accuracy: When training in series on the CAIDA Ark datasets (as shown in Table I), our STL models take $135.4 + 96.15 + 143.4 = 374.95$ seconds on average to train and evaluate all three sub-tasks. In comparison, our MTL approach is able to train and evaluate the same sub-tasks in an average of 15.01 seconds (nearly 96% faster). If the STL models were trained in parallel, the total duration would still be equal to the longest 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 demonstrate equivalent or improved accuracy in comparison to their STL counterparts, with an average F1 score increase of 2.23 percentage points per task as depicted in Figure 3a. The naïve methods (also shown in Figure 3a) depict the accuracy of the statistical classifiers described in § IV-B. These methods are able to identify some portions of loss, noise, and congestion in the measurement data, but are overall largely unable to classify any sub-tasks accurately. Due to their modest-at-best F1 scores, we do not consider these naïve methods for CAIDA's Ark data in the rest of the paper.

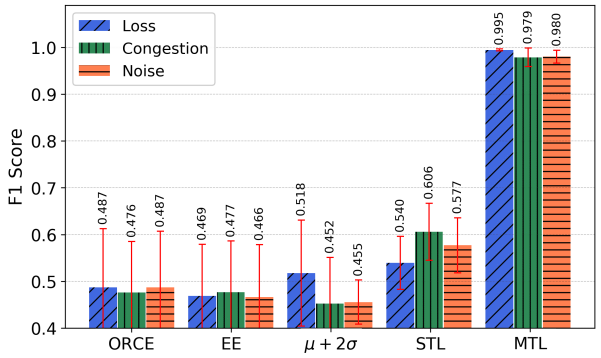
To further generalize our findings, we examine the effects of increasing the quantity of available training data for each of the naïve, STL, and MTL models. To this end, we use the latency data gathered by the RIPE Atlas project [9] and employ the same labeling functions, hyperparameter optimizations, and metrics of analysis as used for the CAIDA datasets, to compare the performances of each approach with respect to the training times and predictive accuracy that each model attains. We also train the same naïve classifiers (ORCE, EE, and $\mu + 2\sigma$) on the RIPE Atlas datasets to ensure that their poor performance on the CAIDA data cannot be attributed to some underlying characteristics in these specific datasets.

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

[5]We utilized the `ping-2021-03-01T0000` dataset from the Atlas daily archives for March 1, 2021, which can be accessed via the RIPE Atlas Measurement Results API [46].



(a) CAIDA Ark Scores



(b) RIPE Atlas Scores

Figure 3: Average F1 score by sub-task comparing naïve methods with both STL and MTL approaches. Higher F1 scores are better.

	Method	Task	Training Time	SD
CAIDA Ark	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
RIPE Atlas	Single Task	LOSS	192.7s	0.865s
		NOISE	197.3s	0.850s
		CONGESTION	203.2s	0.987s
	Multi Task	LOSS NOISE CONGESTION	80.11s	1.22s

Table I: Training time with standard deviations (SDs) for loss, noise, and congestion sub-tasks on the CAIDA Ark and RIPE Atlas datasets. Lower training times are better.

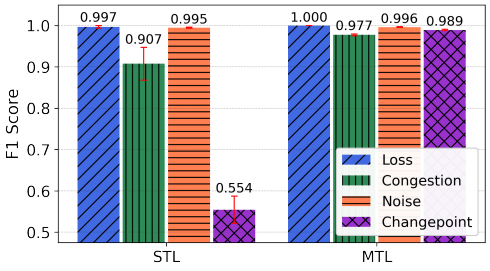
After training the models on the RIPE Atlas data, we find that our multi-task approach adapts well to the new datasets, while the single-task models struggle to perform even the most basic tasks (e.g., LOSS detection). We show the average duration and standard deviation of training times of the three sub-tasks for the RIPE Atlas datasets in Table I. As in the case of the CAIDA datasets, our MTL approach trains much faster than the STL equivalent, training all three sub-tasks on average in 34.90 seconds. In comparison, the STL method takes $192.7 + 197.3 + 203.2 = 593.2$ seconds on average to train all three tasks in series. If the STL approach were parallelized, the training duration would still be equal to the longest training time of the set of tasks (143.4 seconds), which our MTL model still outperforms by nearly 83%.

To complement these results, Figure 3b depicts the F1 scores of each sub-task for the naïve, STL, and MTL approaches on the RIPE Atlas datasets. From this figure, we draw three key observations. First, the MTL model resulting from the ARISE framework outperforms the STL models by over 75%. This demonstrates the fact that ARISE significantly benefits from implicit data augmentation while maintaining model accuracy. Second, we note that the resulting F1 scores are more varied across all STL models, while they are more consistent in our MTL approach. Lastly, we observe that the naïve methods consistently underperform when compared to the STL and MTL models, failing to effectively classify any of the designated sub-tasks. As this behavior has been documented and reflects the findings shown in [4] and [2], we focus only on STL and MTL models for the remaining experiments involving the RIPE Atlas data.

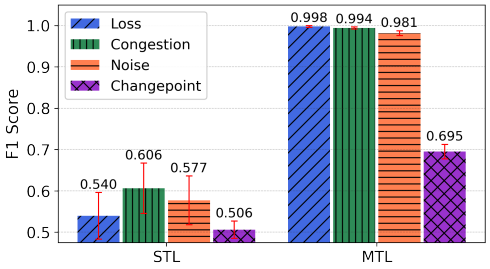
2) Model Scalability: Next, to demonstrate how ARISE facilitates improvements in model scalability, we implement an additional sub-task (CHANGEPOINT detection) in each model pipeline to observe the resulting changes in performance. For CHANGEPOINT classification, our labeling function is as follows: we label a given measurement as a change point if the distance of a measurement to the average of its neighbors exceeds a certain threshold. For the CAIDA datasets (which exhibit a fairly uniform distribution of latencies across all subsets), we select a threshold of 30ms as our changepoint indicator as it appears suitable for the distributions on which we are testing. For the RIPE Atlas datasets, we use $\frac{\sigma}{2}$ (one half of the standard deviation for a given dataset) as our changepoint threshold, as it better accounts for the noisier nature of the underlying distribution of the data.

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 in the STL approach, 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 STL models that suffer 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 re-training both STL and MTL models on the CAIDA datasets, we observe that ARISE-based MTL models outperform the STL approach to an even greater extent than in our initial



(a) CAIDA Ark



(b) RIPE Atlas

Figure 4: Average F1 score by task after the addition of changepoint classification. Higher F1 scores are better.

	Method	Task	Training Time	SD
CAIDA Ark	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
RIPE Atlas	Single Task	LOSS	192.7s	0.865s
		NOISE	197.3s	0.850s
		CONGESTION	203.2s	0.987s
		CHANGEPOINT	208.6s	0.637s
	Multi Task	LOSS NOISE CONGESTION CHANGEPOINT	113.3s	4.00s

Table II: Training times with SDs for all classification sub-tasks on the CAIDA Ark and RIPE Atlas datasets. Lower training times are better.

experiments as Figure 4a shows. More concretely, in adding this sub-task to ARISE, we find that the classification accuracy improves 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 improve slightly (by 0.7% on average), while the average training times increase by only 4.07 seconds (see Table II).

In comparison, the corresponding STL experiments result in no improvement in the performances of other classification tasks, and increase 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 during training, the new task employs the early stopping capabilities of the STL models (described in Appendix C) 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 are shorter, but at the cost of significantly worse classification accuracy. If the STL models were trained concurrently, the training duration would be equal to the length of the longest individual training period across all tasks—in this case, 143.4 seconds, which is still significantly longer than the average training time of our MTL approach.

We also train the STL and MTL models on CHANGEPOINT classification using the RIPE Atlas datasets and the changepoint labeling function mentioned previously. Figure 4 shows the comparison of F1 scores after the addition of CHANGEPOINT classification in the case of RIPE Atlas. We find that our MTL approach continues to outperform against the STL equivalents, but to a less significant degree than in prior experiments. Given the quantity of noise present in the input data, both approaches struggle to properly classify network changepoints to varying extents, with STL attaining an average changepoint classification accuracy of just over 50%, and MTL achieving just under 70%. We show the timing results of training MTL models for RIPE datasets in the lower half of Table II.

D. Meta-Learning vs. MTL

In implementing the additional CHANGEPOINT classification sub-task to examine model scalability as part of ARISE, we observe in § IV-C2 that in the case of the CAIDA Ark dataset, the MTL approach is able to leverage information sharing to allow both the fast adaptation of existing models to new tasks, and the enhancement of other classification tasks as a result of new information being incorporated throughout the model. In this sense, the reported results indicate that MTL provides a way to incorporate *meta-learning*—defined as efficient and fast adaptation to new tasks—in a pragmatic fashion, an effort that prior works such as [5] have endeavored to accomplish theoretically. However, we also notice that in the case of the RIPE Atlas dataset, both STL and MTL models struggle when asked to include the new task.

Motivated by these observations, we choose to also evaluate the efficacy of meta-learning methods such as ensemble learning to perform this CHANGEPOINT classification task. To do so, we feed the predictive output from existing STL models (namely, the noise and congestion classification models) into a new hidden layer acting as a *meta learner* [47] that we then train to perform changepoint classification. In Table III, we show the average F1 scores of each type of model (STL, MTL, and ensemble/meta-learning) across both the CAIDA Ark and RIPE Atlas datasets. We note that while this ensemble model manages to outperform the STL approach, our MTL method outperforms the meta-learning effort by a significant margin.

To explain this result, we note that MTL models leverage similar optimization formulations as some existing meta-learning efforts (such as gradient-based meta-learning [5]). In addition, ARISE already leverages meta-learning techniques

Dataset	STL	Ensemble Model	MTL
CAIDA Ark	0.554	0.686	0.989
RIPE Atlas	0.506	0.578	0.695

Table III: Average F1 Scores for CHANGEPOINT classification in each of the single-task (STL), meta-learning (Ensemble), and multi-task (MTL) approaches.

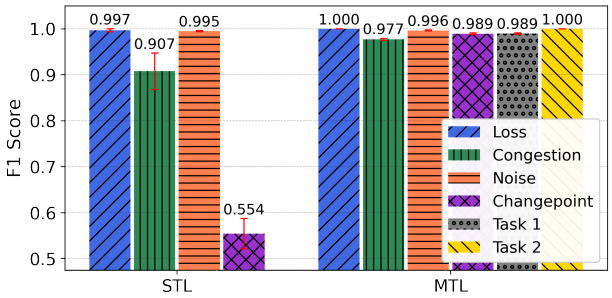
through HPO and information sharing, thus benefitting from both meta- and multi-task learning. As a result, we focus in the remainder of the paper on the comparison between ARISE and the existing STL models.

E. ARISE *in Practice*

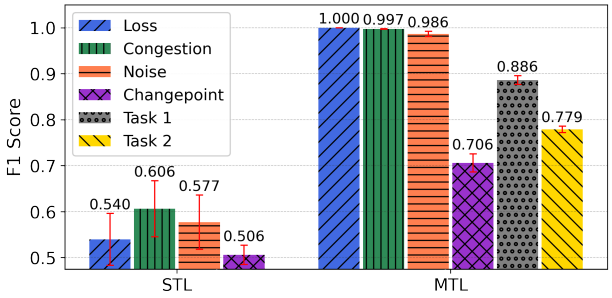
1) Tasks with Multiple Sub-tasks: Beyond isolated feature classification, the modular nature of ARISE is also effective in enabling the construction of complex tasks involving multiple sub-tasks with composite characteristics. To demonstrate this, we combine several task-specific labeling functions to 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 additional 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 = min{NOISE, CHANGEPOINT, CONGESTION}, while TASK 2 = min{NOISE, CHANGEPOINT, LOSS}).

We model **Tasks 1** and **2** (from § I) using ARISE to illustrate how a network operator might interact with our framework. Recall that **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, but 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 ($\vee$) 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.



(a) CAIDA Ark

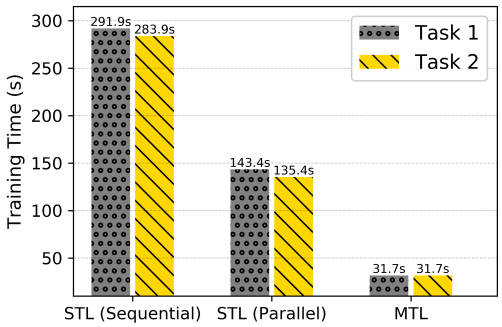


(b) RIPE Atlas

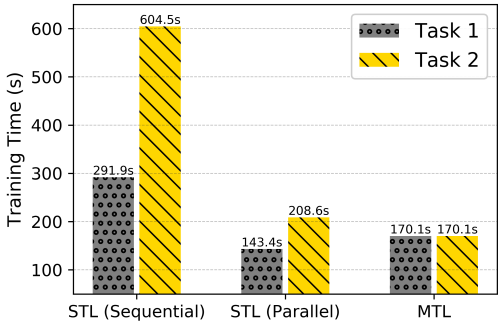
Figure 5: Average F1 scores for all Tasks and sub-tasks. Higher F1 scores are better. The F1 scores for Tasks 1 and 2 in the STL models are equal to the worst-case score among the related sub-tasks.

Figures 5a and 5b show the results of composing sub-tasks for the CAIDA and RIPE datasets, respectively. In Figure 5a, we see that the average F1 scores for Tasks 1 and 2 in the MTL model are 0.989 and 1.0. In contrast, the comparable STL scores are $\min\{0.995, 0.554, 0.907\} = 0.554$ for Task 1 and $\min\{0.995, 0.554, 0.997\} = 0.554$ for Task 2, indicating that our methods significantly outperform the STL equivalent. While the tasks exhibit significant improvements on model accuracies for the CAIDA's datasets, the accuracies of the tasks are lower for the RIPE datasets due to the inherent noise in the data's distribution. The average F1 scores for Tasks 1 and 2 are 0.886 and 0.779 in MTL, compared to the average of 0.506 in STL. Thus, despite the confounding presence of noise in the distribution of data, the MTL accuracy of combined tasks (see Task 1 and Task 2 in Figure 5b) is still significantly better than the equivalent performance in STL.

To complement Figure 5, we show the average training times for both Tasks on each dataset in Figure 6. Note that while the average training time did increase from the previous trial without composite tasks by 12.6 seconds in the CAIDA trials and 56.8 seconds in the RIPE Atlas trials, 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



(a) CAIDA Ark



(b) RIPE Atlas

Figure 6: Average resulting training times for composite Tasks. Lower training times are better.

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 the 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 $96.15 + 52.3 + 135.4 = 283.85$ seconds for Task 2. Similarly, Task 1 would take 609.1 seconds on the RIPE Atlas datasets and Task 2 would take 604.5 seconds. If the STL training were parallelized (meaning each task trained separately but at the same time), 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 Figures 6a and 6b alongside the MTL and sequential STL models. Though this parallelization may result in faster training times for STL models in some cases (i.e., Task 1 in Figure 6b), 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.

2) Task-specific interpretability: Finally, we demonstrate how operators can use ARISE to reason about the decisions made by the resulting MTL models in the context of networking tasks. We note that our discussions regarding interpretability in this section are (a) intentionally qualitative and minimally quantitative, and (b) at the (local) sub-task level rather than (global) task level. There are two main reasons for these limitations. First, our goal is to lower the adoption barrier that efforts like ARISE face today by proposing an initial set of promising methods. Second, creating globally interpretable models is an involved process requiring participation from several stakeholders including network operators, researchers from both the ML and networking communities, and industry practitioners [48]. In what follows, we discuss how ARISE facilitates interpretability at the sub-task level, enabling operators to better trust and understand the decisions made by ML-based tools.

To reason about the decisions made, we encode the operators' domain knowledge in three stages (see Figure 7): the **labeling stage**, or **Stage 1**, (i.e., during the creation and application of labeling functions that are designed to directly capture an operator's know-how in the form of simple programs); the **composition stage**, or **Stage 2**, (i.e., during the identification of composite characteristics across sub-tasks and training of shared layers by using those characteristics); and the **production stage**, or **Stage 3**, (i.e., how operators leverage relationships between sub-tasks to inform their understanding of a model's decision-making procedures and consequently improve upon the design in stages 1 and 2).

In Stage 1, operators can employ their heuristic-based labeling functions to generate weak labels for all data points within a dataset. However, these labels do not necessarily reflect the true categorical classifications of all data points. To illustrate, consider a subset of latency data from CAIDA's Ark project. The noise and congestion thresholds generated by the labeling functions (described in § III-B) overlaid on top of this subset are shown in Figure 8. These thresholds represent the values an operator could use to classify measurements as instances of either noise or congestion—but how do we distinguish between the two? Said differently, operators can loosely classify entire datasets but possess no insight into the reasoning of classification decisions beyond their initial assumptions regarding the data being used. Consequently, operators cannot accurately infer the distribution of network features they wish to classify to any greater extent than that provided by their original heuristics. In the context of Figure 8, this suggests that operators in Stage 1 are only able to qualify extreme measurements such as those high enough to be considered only noise/congestion, or those low enough to be considered neither noise nor congestion. The isolated use of basic labeling functions yields crude results that are particularly ineffective in distinguishing network features that exhibit similar characteristics (e.g., the measurement entries near the labeling thresholds in Figure 8 which cannot be easily distinguished as either noise or congestion).

In Figure 9, we show the crude classification of measurement data points (Figure 8) at Stage 1, where the percentage values associated with each box represent the approximate

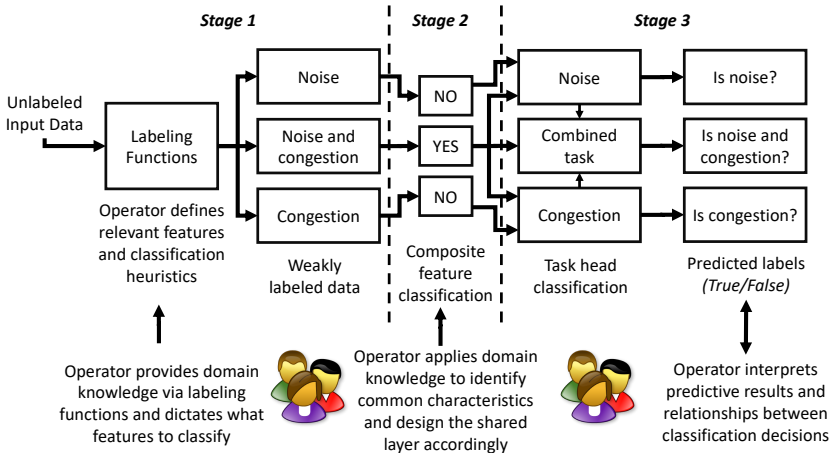


Figure 7: Task-specific interpretability stages in ARISE.

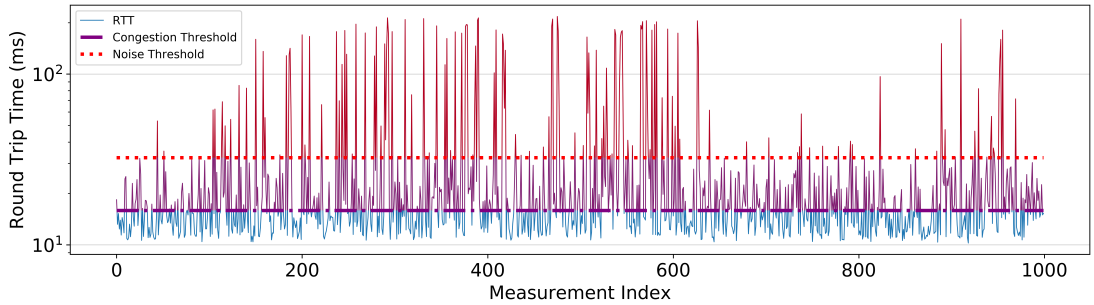


Figure 8: Applying labeling functions to raw data. Any measurements above the noise threshold (red) are labeled as noise, while any measurements between the noise and congestion threshold (purple) are labeled as congestion. Measurements in blue are good/benign (i.e. neither noise, nor congestion).

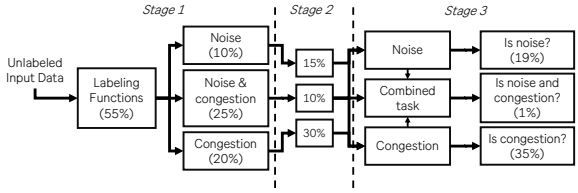


Figure 9: Feature classification by interpretability stage in ARISE using data from Figure 8. ∼55% measurements were labeled as either noise or congestion while the remainder were unlabeled or "good" measurements such as those below the congestion threshold in Figure 8.

percentage of measurements in our subset of data that were labeled positively. The individual noise and congestion boxes reflect the percentage of measurements that were labeled with relative certainty, whereas the combined 'Noise & Congestion' box represents the percentage of measurements that fell within an area of uncertainty while applying our labeling functions. At the end of Stage 1, a majority (approximately 25%) of the features classified by ARISE are contained within the uncertain 'Noise & Congestion' category. To reduce this uncertainty and improve our classification accuracy, we then move to leverage the common characteristics across sub-tasks in ARISE's shared layers.

In Stage 2, operators can begin leveraging task commonalities during the training process to restructure ARISE's shared layers and inform its classification decisions. In identifying the relationships between noise and congestion (e.g., measurement volatility as described in § II), operators can begin to distinguish between measurements that fall into areas of uncertainty regarding multiple feature classification categories (e.g., latency measurements surrounding the noise and congestion thresholds in Figure 8), allowing them to refine the boundaries of their labeling functions and develop a stronger intuition regarding ARISE's classification procedures. We see this in action in Stage 2 of Figure 9, where ARISE's information sharing capabilities enable us to identify some features that were misclassified in Stage 1 as a result of the initial reliance on naïve labeling functions. Consequently, the uncertainty of classifications in Stage 2 is dramatically reduced (dropping from 25% to 10%) while each of the noise and congestion-specific labels grow more confident in their classification accuracy. However, ARISE's information sharing abilities alone are not enough to eradicate uncertainty in feature classification, as the complex relationships between distinct sub-tasks may not be automatically captured by ARISE's training procedures, or new relationships may arise that network operators had not previously anticipated. Thus, we turn to operator expertise to further refine the information sharing and interpretability components of ARISE.

In Stage 3, operators employ ARISE to its fullest extent by engaging their domain expertise with the results of classification predictions across sub-tasks to then infer causality behind specific decisions. After training the model in ARISE, an operator can then evaluate its predictions on sample data to identify, isolate, and further leverage the relationships that arise between network features in refining ARISE's shared lay-

ers and information-sharing capabilities. This allows operators to identify network features from measurement data efficiently with a high rate of accuracy while also enabling them to interpret the reasoning behind ARISE's classification decisions. Thus, operators can accurately interpret the reasoning behind ARISE's policies in nearly every classification decision and can further improve the explainability of questionable decisions by implementing additional classification tasks to isolate inter-task relationships and leverage operator expertise. In the context of Figure 9, this means that Stage 3 of ARISE can provide network operators with significant improvements to classification confidence, as the uncertainty of noise vs. congestion classification falls to its lowest rate among the three stages while task-specific classification confidence reaches its peak. Thus, we observe that ARISE excels in enhancing the local interpretability of ML models and classification systems and provides a meaningful avenue to improve operator trust in, and understanding of, the ML management tools they employ in their networks. We leave the further exploration and refinement of model interpretability to future work.

V. DISCUSSION

Implications of ARISE. Our findings suggest that multi-task learning models informed by meta-learning techniques have the ability to significantly improve the applicability of machine learning in the networking domain. Our experiments indicate that MTL typically outperforms prior efforts in both single-task and meta-learning from an empirical and theoretical perspective, resulting simultaneously in enhanced model performance, shorter training times, and the added benefits of providing additional insight into the reasoning behind its classification decisions. As a result, frameworks such as ARISE have the potential of allowing operators to leverage the full power of ML at scale in their networks by overcoming the limitations of prior efforts and providing insight into the reasoning behind their classification decisions.

Using ARISE for new classification tasks. Researchers and/or network operators may wish to use ARISE to generate a multi-task model for networking tasks other than the ones considered in this paper. The following describes a step-by-step approach for doing so, and we illustrate the different steps with the example of an amplification-style DDoS attack detection task.

- Identify a data source (e.g., NetFlow) and generate the time series for the network features of interest such as *heavy hitters* (e.g., in terms of number of flows per second, unique number of source IP addresses, etc.).
- Translate the available domain heuristics into concrete labeling functions that can be applied as code. For example, instances where the number of flows per second $> A$ and the number of distinct sources $> B$ are labeled as indications of the onset of a DDoS attack.
- Construct the task heads and shared layers, employing either complete information sharing (as discussed in § III-C) or some alternative information sharing paradigm. For example, if all attacks share certain heavy hitters-based features, then these features can be used to create shared layers. At the same time, if the attacks differ in their protocol-specific features, then those features can be used to train attack-specific head layers (e.g., ICMP vs. NTP vs. DNS).
- Train, test, modify, and refine the model's hyperparameters using either HPO or other elements of domain expertise.

Limitations of ARISE. ARISE is designed to facilitate easy-adoption in different applied settings, and the considered labeling functions illustrate this design objective. However, adapting the model structure and underlying architecture of the shared and head layers is a challenge that depends greatly on both the features being classified and the datasets being used. Furthermore, the interpretative capabilities of ARISE depend on the abilities of the network operators employing the framework, as the operators must be able to leverage their domain knowledge to effectively intuit the reasoning or causality behind certain classification decisions. Thus, while ARISE shows promise in enabling networking domain experts to begin reasoning about the decisions made by their tailor-made classification models, such interpretative capabilities are not yet accessible to all users such as those with less relevant domain expertise.

VI. SUMMARY AND FUTURE WORK

The use of ML for networking faces several obstacles including a lack of labeled datasets, scarcity of operators' domain expertise during the labeling process, multiplicative model training times with increasing number of network tasks, and a general inability to reason about the decisions made by the trained models. In this paper we propose ARISE, a novel framework to overcome those obstacles. ARISE incorporates operator expertise in the form of labeling functions; combines innovations in weak supervision, multi-task learning, and meta-learning to address prior limitations regarding model scalability, long training times, etc.; and enables the adoption of ML for networking by democratizating the pipeline with task-specific interpretability.

We demonstrate the efficacy of ARISE by using two large datsets of real-world network measurements, considering different tasks such as inferring loss, congestion, change points, and noise, and comparing the performance of ARISE-based MTL vs. STL models. 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 prior STL efforts. Furthermore, our experiments suggest that MTL models aided by meta-learning techniques (e.g., hyperparameter optimization) can outperform explicit meta-learning methods such as ensemble learning. We also report findings that show the promise of MTL models for providing a means for reasoning about their decision-making process, at least at the level of individual tasks.

As part of our future work, we plan to expand the capabilities of ARISE in three different directions. The first direction concerns using ARISE to rethink cyber attack detection tasks in terms of a hierarchical structure based on attack goals, vulnerabilities exploited, and other domain-specific aspects. This way, instead of building flat multi-class models to detect all possible attacks or building several specialized task-specific

models, attack detection models could be built based on hierarchical relationships between attacks, enabling unprecedented information sharing throughout the hierarchy. The second direction focuses on leveraging an ARISE-like framework to drastically transform current network management practices through the use of a *conversational intelligence approach* whereby operators will be able to "tell" the network what to do by directly "talking" to an intelligent network management agent. 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 [49], [50] and demonstrating its feasibility with a spectrum of illustrative network management task examples. Finally, to further increase operators' trust in trained learning models, we plan to enhance the robustness of ARISE with adversarial examples—intentional feature perturbations that can cause a model to make false predictions.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their insightful feedback. This work is supported by the National Science Foundation through CNS 1850297, CNS 2145813, and OAC 2126281, and a UO 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 UO.

REFERENCES

[1] N. Feamster and J. Rexford, "Workshop on self-driving networks." https://nsf-srn-2018.cs.princeton.edu/nsf-srn-report.pdf, 2018.

[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*, NetAI '20, (New York, NY, USA), p. 21–27, Association for Computing Machinery, 2020.

[3] K. Claffy, D. Clark, J. Heidemann, F. Bustamante, M. Jonker, A. Schulman, and E. Zegura, "Workshop on Overcoming Measurement Barriers to Internet Research (WOMBIR 2021) Final Report," vol. 51, p. 33–40, July 2021.

[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)*, pp. 479–484, IEEE, 2019.

[5] H. Wang, H. Zhao, and B. Li, "Bridging Multi-Task Learning and Meta-Learning: Towards Efficient Training and Effective Adaptation," 2021.

[6] 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*, pp. 93–100, 2019.

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

[8] "CAIDA Ark Datasets." www.caida.org/projects/ark/topo_datasets.xml.

[9] "RIPE Atlas." https://atlas.ripe.net, 2018.

[10] K. Xu and J. Chandrashekar and Z.L. Zhang, "A First Step toward Understanding Inter-domain Routing Dynamics," in *ACM SIGCOMM workshop on Mining network data*, 2005.

[11] 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.

[12] 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.

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

[14] G. Comarela, R. Durairajan, P. Barford, D. Christenson, and M. Crovella, "Assessing Candidate Preference through Web Browsing History," in *proceedings of ACM SIGKDD*, 2018.

[15] D. Wang, Y. Yu, and A. Rinaldo, "Optimal change point detection and localization in sparse dynamic networks," *The Annals of Statistics*, vol. 49, pp. 203–232, Feb. 2021.

[16] H. Ringberg and A. Soule and J. Rexford and C. Diot, "Sensitivity of PCA for Traffic Anomaly Detection," *SIGMETRICS*, 2007.

[17] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing Network-wide Traffic Anomalies," in *ACM SIGCOMM*, 2004.

[18] A. Lakhina, M. Crovella, and C. Diot, "Mining Anomalies Using Traffic Feature Distributions," *ACM SIGCOMM*, 2005.

[19] J. Camacho, A. Pérez-Villegas, P. García-Teodoro, and G. Maciá-Fernández, "PCA-based Multivariate Statistical Network Monitoring for Anomaly Detection," *Computers & Security*, 2016.

[20] X. Li, F. Bian, H. Zhang, C. Diot, R. Govindan, W. Hong, and G. Iannaccone, "MIND: A Distributed Multi-Dimensional Indexing System for Network Diagnosis," in *IEEE INFOCOM*, 2006.

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

[22] A. W. Moore and D. Zuev, "Internet traffic classification using Bayesian analysis techniques," in *ACM SIGMETRICS*, 2005.

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

[24] M.-C. Yuen, I. King, and K.-S. Leung, "A survey of crowdsourcing systems," in *IEEE SocialCom*, 2011.

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

[26] A. J. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré, "Snorkel: Rapid training data creation with weak supervision," *VLDB Endowment*, 2017.

[27] 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*, pp. 3567–3575, 2016.

[28] 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.

[29] 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.), pp. 4765–4774, 2017.

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

[31] B. Li, Y. Lin, and S. Zhang, "Multi-Task Learning for Intrusion Detection on Web Logs," *J. Syst. Archit.*, vol. 81, p. 92–100, nov 2017.

[32] J. Yang, X. Sun, Y.-K. Lai, L. Zheng, and M.-M. Cheng, "Recognition from Web data: A progressive filtering approach," *IEEE Transactions on Image Processing*, vol. 27, no. 11, pp. 5303–5315, 2018.

[33] S. Bickel, J. Bogojeska, T. Lengauer, and T. Scheffer, "Multi-Task Learning for HIV Therapy Screening," in *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, (New York, NY, USA), p. 56–63, Association for Computing Machinery, 2008.

[34] L. Weng, "Meta-Learning: Learning to Learn Fast," *lilianweng.github.io*, 2018.

[35] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[36] D. J. Rezende, S. Mohamed, I. Danihelka, K. Gregor, and D. Wierstra, "One-Shot Generalization in Deep Generative Models," 2016.

[37] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-Learning with Memory-Augmented Neural Networks," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, p. 1842–1850, JMLR.org, 2016.

[38] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[39] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[40] F. Hutter, L. Kotthoff, and J. Vanschoren, eds., *Automated Machine Learning - Methods, Systems, Challenges*. Springer, 2019.

[41] "Scamper." https://www.caida.org/tools/measurement/scamper/.

[42] "tsfresh." https://tsfresh.readthedocs.io/en/latest/, 2019.

[43] "Multi-Task Learning (MTL) Basics." https://www.snorkel.org/use-cases/multitask-tutorial, 2019.

[44] A. Ratner, B. Hancock, J. Dunnmon, F. Sala, S. Pandey, and C. Ré, "Training Complex Models with Multi-Task Weak Supervision," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, p. 4763–4771, 2019.

[45] "Cloudlab.us." https://www.cloudlab.us/, 2021.
[46] "RIPE Atlas Measurement Results API." https://beta-docs.atlas.ripe.net/apis/measurement-results/measurement-results.html.
[47] V. Pimpalkhute, A. Pandit, M. Mishra, and R. Singhal, "Accelerating Gradient-based Meta Learner," *CoRR*, vol. abs/2110.14459, 2021.
[48] H. Bastani, O. Bastani, and C. Kim, "Interpreting predictive models for human-in-the-loop analytics," *arXiv preprint arXiv:1705.08504*, pp. 1–45, 2018.
[49] A. Alsudais and E. Keller, "Hey Network, Can You Understand Me?," in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 193–198, IEEE, 2017.
[50] 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)*, pp. 609–623, 2018.
[51] C. Truong, L. Oudre, and N. Vayatis, "Selective review of offline change point detection methods," *Signal Processing*, vol. 167, p. 107299, 2020.
[52] P. Fryzlewicz, "Wild binary segmentation for multiple change-point detection," *The Annals of Statistics*, vol. 42, Dec 2014.

APPENDIX

A SOURCE CODE

The source code and the datasets used in this work can be found at https://gitlab.com/onrg/arise.

B CHANGEPOINT CLASSIFICATION

The *ruptures* [51] 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. As a baseline, we sample 1,000 values from a uniform distribution containing three changepoints at indices 250, 500, and 750 of the time-series. We then employ the binary segmentation [52] 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.

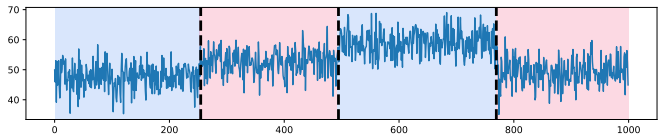


Figure 10: Using *ruptures* to classify known changepoints. True changepoints are shown by the transitions in background color, predicted changepoints are shown via the dotted lines.

As Figure 10 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 is not known or only a portion of the time-series is available on which to classify (as is often the case when working with 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.

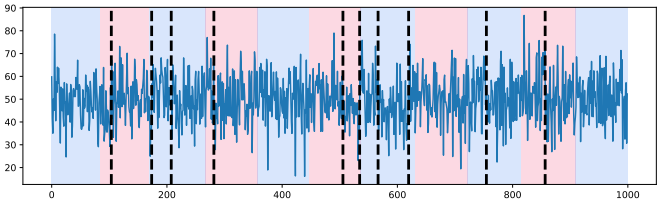


Figure 11: Classifying an unknown quantity of changepoints using window-sliding segmentation. True changepoints are shown by the transitions in background color, predicted changepoints are shown via the dotted lines.

As Figure 11 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.

C ML 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 an operator may wish to implement. To select these hyperparameters, we initially attempted to manually iterate across several possible values at different scales to identify an optimal set of parameters, but quickly found that this approach was not practical in the context of the model and datasets we employ. For one, despite our best efforts, attempting to train our models with hyperparameters chosen by naïve selection failed to result in satisfactory model performance with respect to the F1 score when evaluated on the withheld testing set. In addition, we also found that the optimal parameterization for a given model was dependent on the specific dataset on which it was trained. As our analysis of ARISE relies on evaluating its performance across multiple distinct datasets and the amount of time it takes to train effectively on said datasets, we concluded that such an approach would be ineffective for the context of our experiments (especially when used in the context of real-world applications). Instead, we chose to leverage insight from meta-learning techniques such as AutoML [40] to quickly iterate across many different potential hyperparameter values and select the resulting values that perform best on the dataset and tasks at hand. By automating

Hidden Layers	F1 Score	Training Time	SD
2 (base)	0.554 ± 0.03286	52.30s	5.84s
3	0.565 ± 0.03418	134.43s	27.15s
5	0.561 ± 0.03458	145.60s	26.26s
10	0.560 ± 0.03244	177.65s	37.45s

Table IV: Additional training experiments conducted on the CHANGEPOINT classification sub-task for the CAIDA Ark datasets depicting the impact of increasing the number of hidden layers in the STL models. F1 score is $\pm$ the Standard Error.

this process, we were able to evaluate a significantly larger portion of the hyperparameter space in a fraction of the time, and thus improve the performance of both the STL and MTL models as a result.

From here, we then evaluate other HPO strategies such as learning rate schedulers that linearly decrease the learning rate upon subsequent training iterations, which we found to be particularly effective in our ARISE-based models as it allows the MTL models to remain more accurate and consistent in their resulting classification capabilities. We found that while some hyperparameters (primarily the model learning rates) required substantial optimization, others were less dependent on the dataset or tasks at hand and were thus able to remain fixed for the duration of our experiments (e.g., the number of training epochs was set to 20 for the STL models and 10 for the MTL models, as increasing the number of epochs beyond these values yielded little to no improvement in terms of the model accuracy and often resulted in extended training times).

In addition to optimizing the hyperparameters of each model, we also evaluate the effects of increasing the number of hidden layers in the STL models beyond the baseline of two hidden layers established in [2]. While we found a depth of two hidden layers to be effective for each of the LOSS, NOISE, and CONGESTION sub-tasks in the CAIDA datasets, we found that the STL models were less performant when considering the CHANGEPOINT sub-task or when trained on the RIPE Atlas datasets. To determine whether this lack of performance is due to an insufficient number of hidden layers in the STL models, we evaluate the effect of increasing the number of hidden layers from two to three, five, and ten hidden layers by adding in additional dense layers and retraining each of the STL models. We show the results of these experiments in Table IV for the CHANGEPOINT sub-task on the CAIDA datasets and note that retraining the STL models with additional hidden layers for the LOSS, NOISE, and CONGESTION sub-tasks in the RIPE Atlas datasets yielded similar results. As incorporating additional hidden layers appears to provide no significant benefit to the STL models' F1 scores, we elect to retain our initial baseline of two hidden layers as it achieves near equivalent performance to each of the other model depths in the least amount of time.

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. Given the number of training datasets from RIPE and CAIDA, we calculate and report the average of F1 scores across those datasets. 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 and data pre-processing 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 also leverage a data augmentation strategy similar to that described in [2], where additional examples of each network feature are synthesized and inserted into the training sets to provide the STL models with more examples to learn from. This helps to ensure both that the STL models have enough examples of the features they are learning to identify, and make them more robust in adapting to more widely varied datasets. We choose 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.

Jared Knofczynski is a Ph.D. student in the Department of Computer Science at the University of Oregon. His work has been recognized with several awards, including the Phillip Seeley Graduate Fellowship, UO VPRI Fellowship, Ripple UBRI Scholarship, and Jean Wittemyer Memorial Scholarship.

Ramakrishnan Durairajan is an Assistant Professor in the Department of Computer Science at the University of Oregon. His research has been recognized with NSF CAREER, NSF CRII, Ripple faculty fellowship, UO faculty research award, several best paper awards, and has been covered in several fora.

Walter Willinger is Chief Scientist at NIKSUN, Inc. Before joining NIKSUN, he worked at AT&T Labs-Research and at Bellcore Applied Research. He is co-recipient of the 1995 W.R. Bennett Prize Paper Award, the 1996 W.R.G. Baker Prize Award, and of the 2005 and 2016 ACM/SIGCOMM Test-of-Time Paper Awards.